STACKSIMPLIFY

GCP Terraform on Google Cloud: DevOps SRE 30 Real-World Demos

Kalyan Reddy Daida

Terraform on Google Cloud

STACKSIMPLIFY

Terraform Fundamentals

Terraform CLI Install	1
Terraform Commands	2
Terraform Language Basics	3
Meta-Argument Provider	4
Input Variables & Output Values	5
Meta-Argument count	6
Terraform Data sources	7
Meta-Argument for_each	8
Terraform Local Values	9

30+
Real-World Demos

Incremental approach to build complex infra

Step by Step GitHub Documentation

GCP Services

10	GCP Managed Instance Groups
11	GCP Regional Application Load Balancer HTTP
12	MIG Private IP with Cloud NAT and Cloud Router
13	GCP MIG Update Policy
14	GCP Certificate Manager with self-signed SSL
15	GCP Cloud Domains and Cloud DNS
16	GCP Cloud DNS + Prod grade SSL with Certificate Manager
17	GCP ALB Context Path Routing
18	GCP ALB Domain Name based Routing

Terraform on Google Cloud

STACKSIMPLIFY

GCP Services	
GCP ALB Header based Routing	19
GCP Cloud Logging (Send App logs to Cloud Logging)	20
GCP Cloud Monitoring (Uptime Checks, Alert Policy)	21
GCP Cloud SQL with Public IP	22
DNS to DB with Self-signed SSL + Cloud SQL Public IP	23
DNS to DB with Cloud DNS + Cloud SQL Public IP	24
GCP Cloud SQL with Private IP	25
DNS to DB with Self-signed SSL + Cloud SQL Private IP	26
DNS to DB with Cloud DNS + Cloud SQL Private IP	27

30+
Real-World Demos

Incremental way to build complex infra

Step by Step GitHub Documentation

GCP Services	
28	Terraform Modules from Public Registry
29	Build Terraform Custom Module
30	GCP DevOps for Terraform Configs: Cloud Build & GitHub
31	GCP Global Application Load Balancer

Terraform on Google Cloud

STACKSIMPLIFY

Google Compute Engine

Virtual Private Cloud

Cloud Monitoring

Cloud SQL (MySQL)

Cloud Load Balancing

Cloud Logging

Cloud IAM

Cloud Storage

Cloud Firewalls

Cloud Domains

Cloud DNS

Cloud External IP

Cloud Router

Cloud NAT

Cloud Build

Persistent Disks

Certificate Manager

17+ GCP Services

Many more **Sub-services** under each service

Automated with Terraform

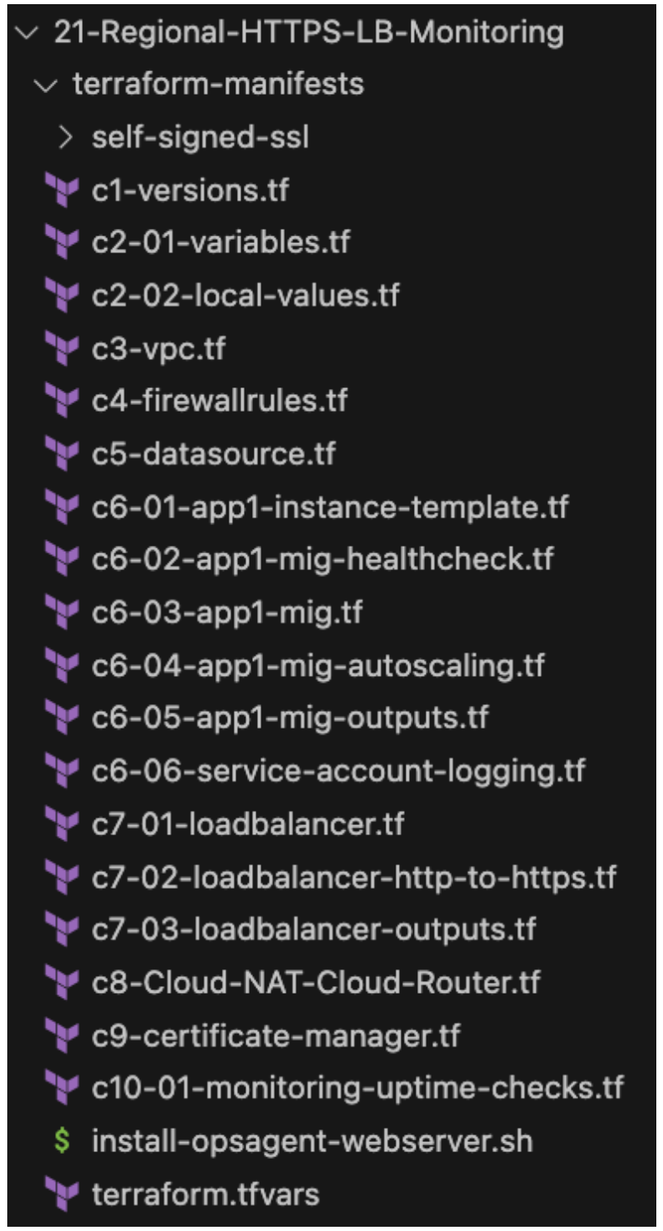
GitHub Step-by-Step Documentation

STACKSIMPLIFY

- terraform-on-google-cloud
  - 01-Terraform-Install-Tools
  - 02-Terraform-Commands
  - 03-Terraform-Language-Basics
  - 04-Terraform-MetaArgument-provider
  - 05-Terraform-Variables-Output-Values
  - 06-Terraform-MetaArgument-count
  - 07-Terraform-Datasources
  - 08-Terraform-MetaArgument-foreach
  - 09-Instance-Templates-and-LocalValues
  - 10-Managed-Instance-Groups-MIGPublicIPs
  - 11-Regional-HTTP-LB-MIGPublic
  - 12-Regional-HTTP-LB-MIGPrivate
  - 13-Regional-HTTP-LB-MIGUpdatePolicy
  - 14-Regional-HTTPS-LB-SelfSigned
  - 15-Cloud-Domains-and-Cloud-DNS
  - 16-Regional-HTTPS-LB-CloudDNS
  - 17-Regional-HTTP-LB-PATH-Routing
  - 18-Regional-HTTP-LB-HOST-Routing
  - 19-Regional-HTTP-LB-HEADER-Routing
  - 20-Regional-HTTPS-LB-Logging
  - 21-Regional-HTTPS-LB-Monitoring
  - 22-CloudSQL-PublicDB-TF-Remote-State
  - 23-DNS-to-DB-SelfSigned-CloudSQL-PublicDB
  - 24-DNS-to-DB-CloudDNS-CloudSQL-PublicDB
  - 25-CloudSQL-PrivateDB
  - 26-DNS-to-DB-SelfSigned-CloudSQL-PrivateDB
  - 27-DNS-to-DB-CloudDNS-CloudSQL-PrivateDB
  - 28-Terraform-Modules
  - 29-Terraform-Build-Custom-Module
  - 30-Terraform-GCP-DevOps-CloudBuild-GitHub
  - 31-Global-HTTP-LB-MIGPrivate

How are Terraform Configs organized ?

STACKSIMPLIFY



Every GCP resource is organized in an incremental manner with file number c1, c2, c3 for easy understanding, reading and troubleshooting

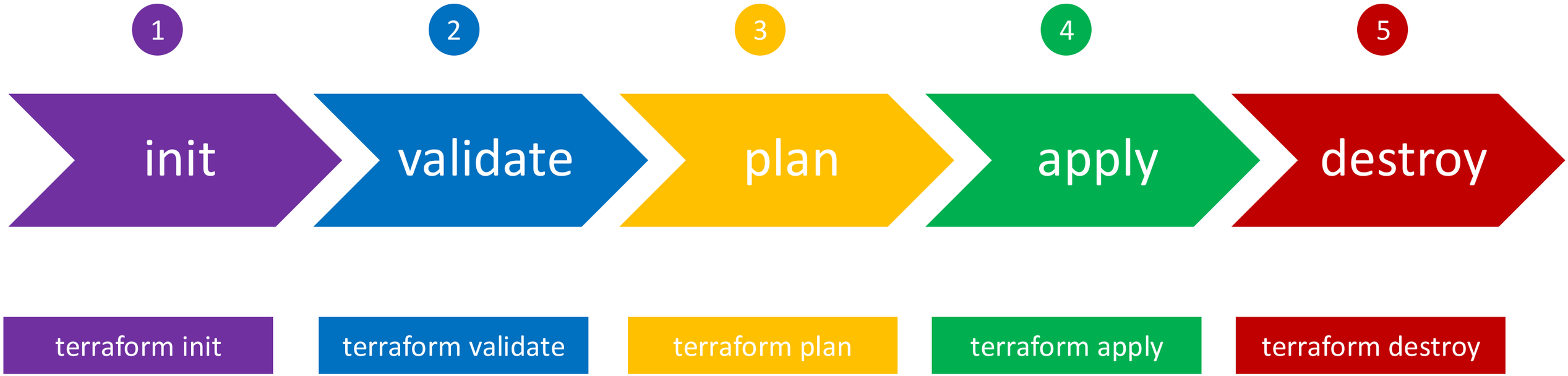
Every section or demo is well organized in GitHub and contains step by step documentation

GitHub Repository

Repository Used For	Repository URL
Course Main Repository with step-by-step documentation	https://github.com/stacksimplify/terraform-on-google-cloud
GCP DevOps for Terraform Configs using Cloud Build and GitHub	https://github.com/stacksimplify/gcp-terraform-devops

STACKSIMPLIFY

Terraform Workflow



Terraform language uses a limited number of **top-level block** types, which are **blocks** that can appear outside of any other **block** in a TF configuration file.

Terraform Top-Level Blocks

Most of **Terraform's features** are implemented as **top-level** blocks.

- Terraform Block
- Providers Block
- Resources Block

Fundamental Blocks

- Input Variables Block
- Output Values Block
- Local Values Block

Variable Blocks

- Data Sources Block
- Modules Block

Calling / Referencing Blocks

- Import Block
- Moved Block
- Removed Block
- Check Block

NEW BLOCKS – Added Recently

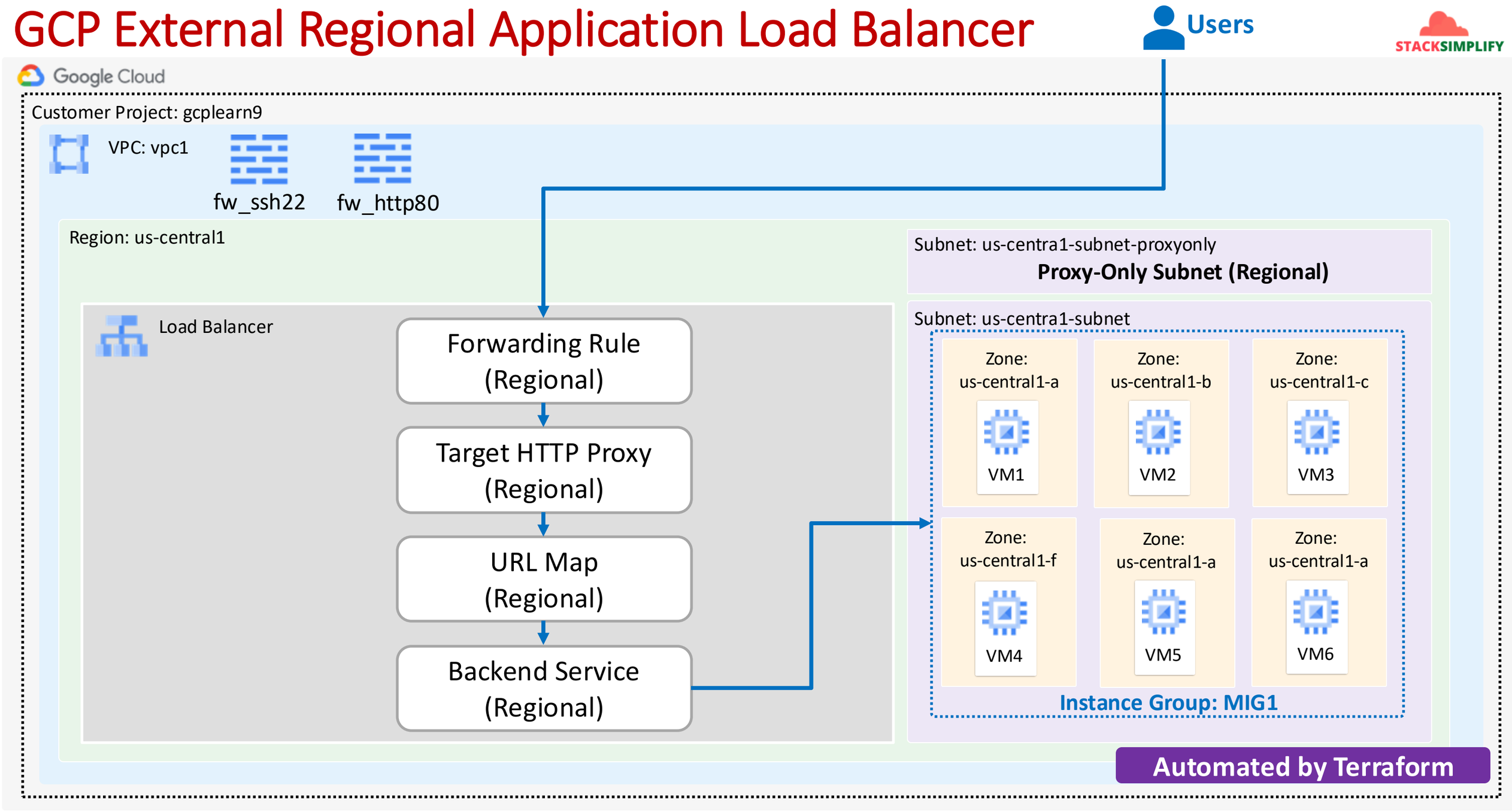
Terraform Fundamentals

STACKSIMPLIFY

Demo-01: Install CLI Tools: gcloud CLI, Terraform CLI, VSCode Editor	7 lectures • 26min
Demo-02: Terraform Commands (init, validate, plan, apply and destroy)	3 lectures • 27min
Demo-03: Terraform Language Basics	10 lectures • 1hr 15min
Demo-04: Terraform Meta-Argument - Provider (Multiple Providers Demo)	2 lectures • 11min
Demo-05: Terraform Input Variables and Output Values	5 lectures • 38min
Demo-06: Terraform Meta-argument: count	4 lectures • 28min
Demo-07: Terraform Datasources	2 lectures • 11min
Demo-08: Terraform Meta-argument: for_each	3 lectures • 20min
Demo-09: Terraform Local Values, GCP Instance Templates	5 lectures • 36min

Terraform Fundamentals
(4.5 Hours)

GCP External Regional Application Load Balancer

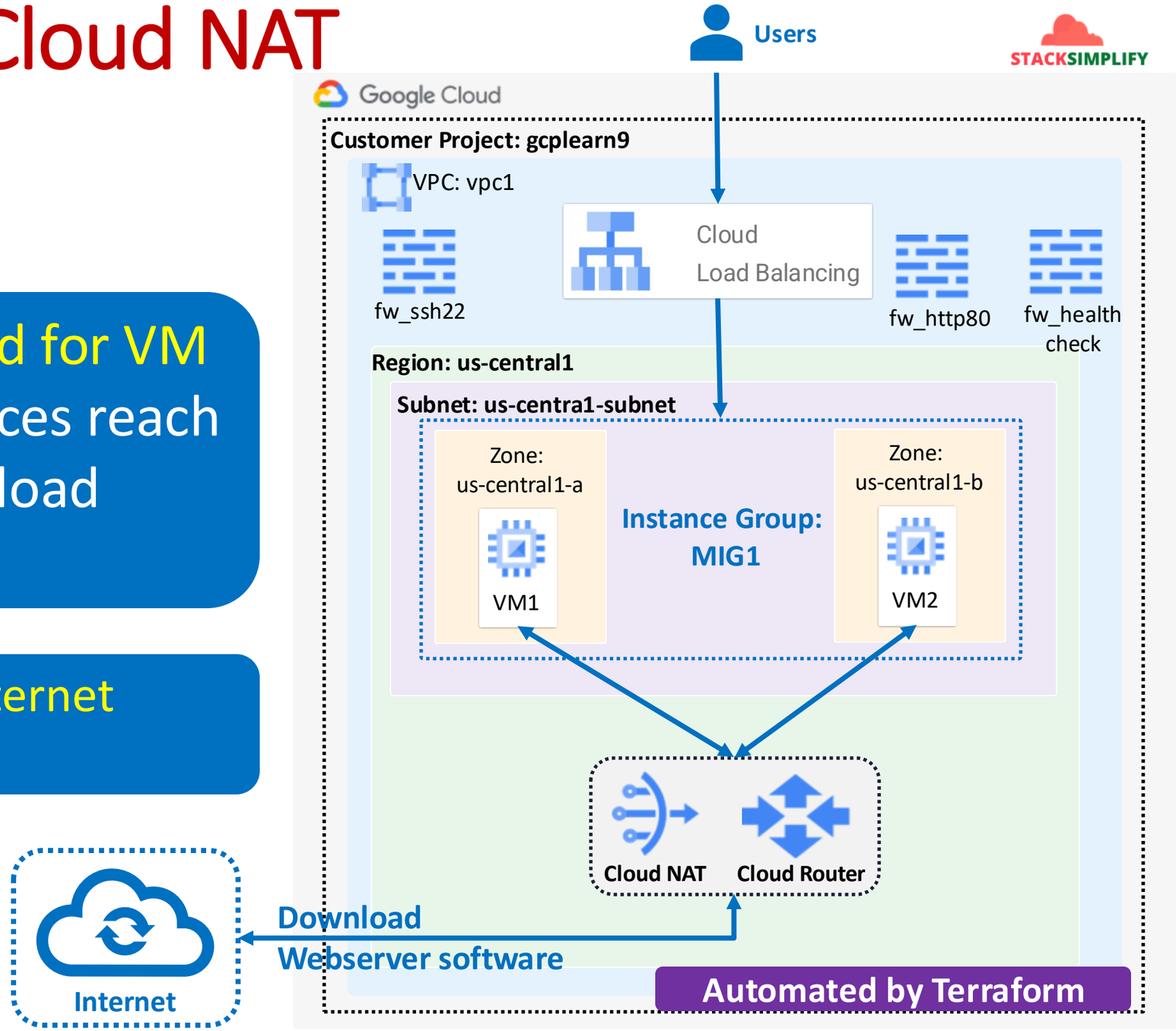


GCP Cloud Router + Cloud NAT

Use case

When external IPs are disabled for VM instances, how will VM instances reach the external internet to download software (e.g., Nginx binary)?

How will you open an outbound internet connection for VM instances?

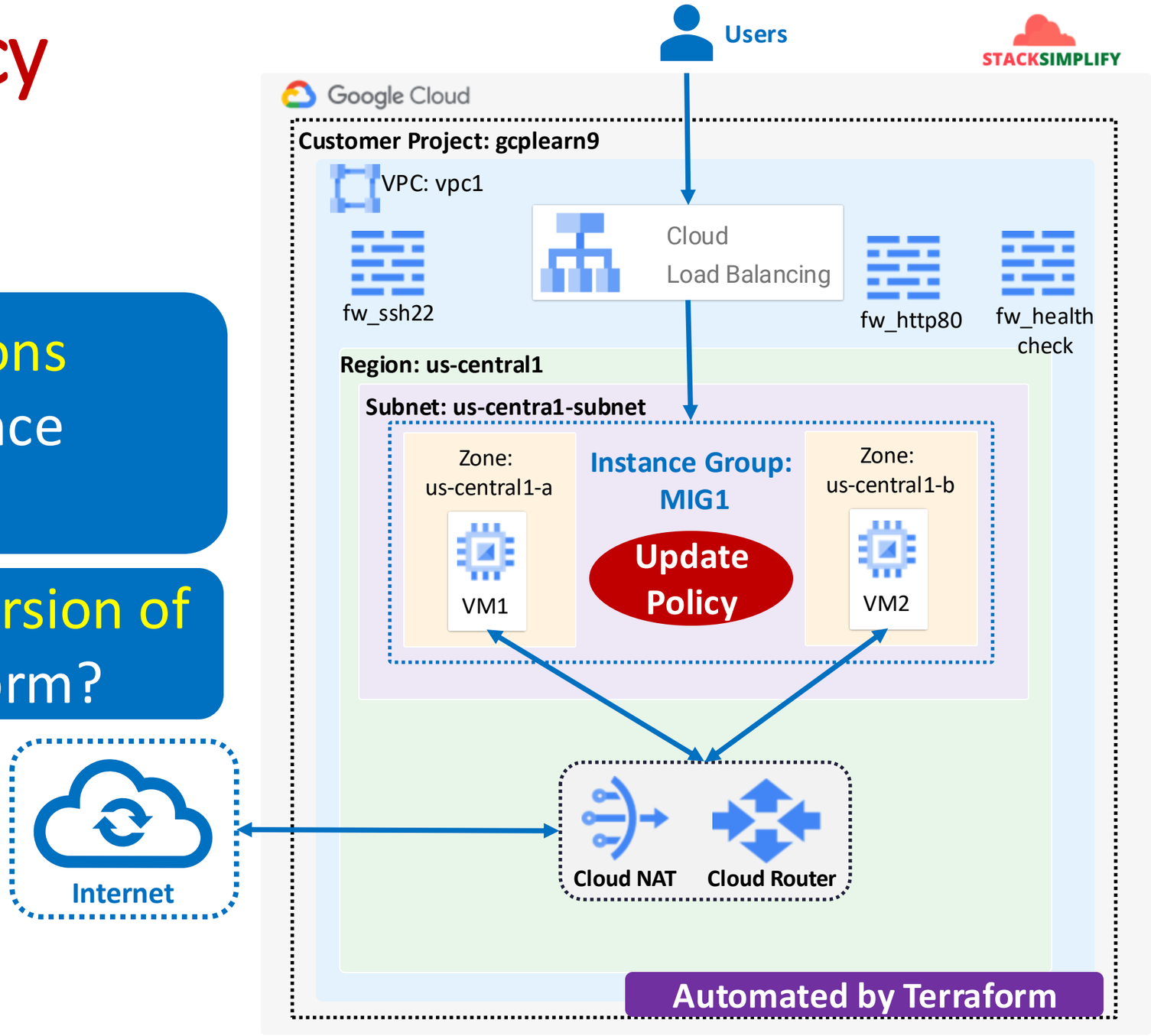


GCP MIG Update Policy

Use case

How will you update applications hosted in GCP Managed Instance Groups using Terraform?

How do you upgrade the V1 version of app to V2 version using Terraform?



GCP Load Balancer Self-signed SSL

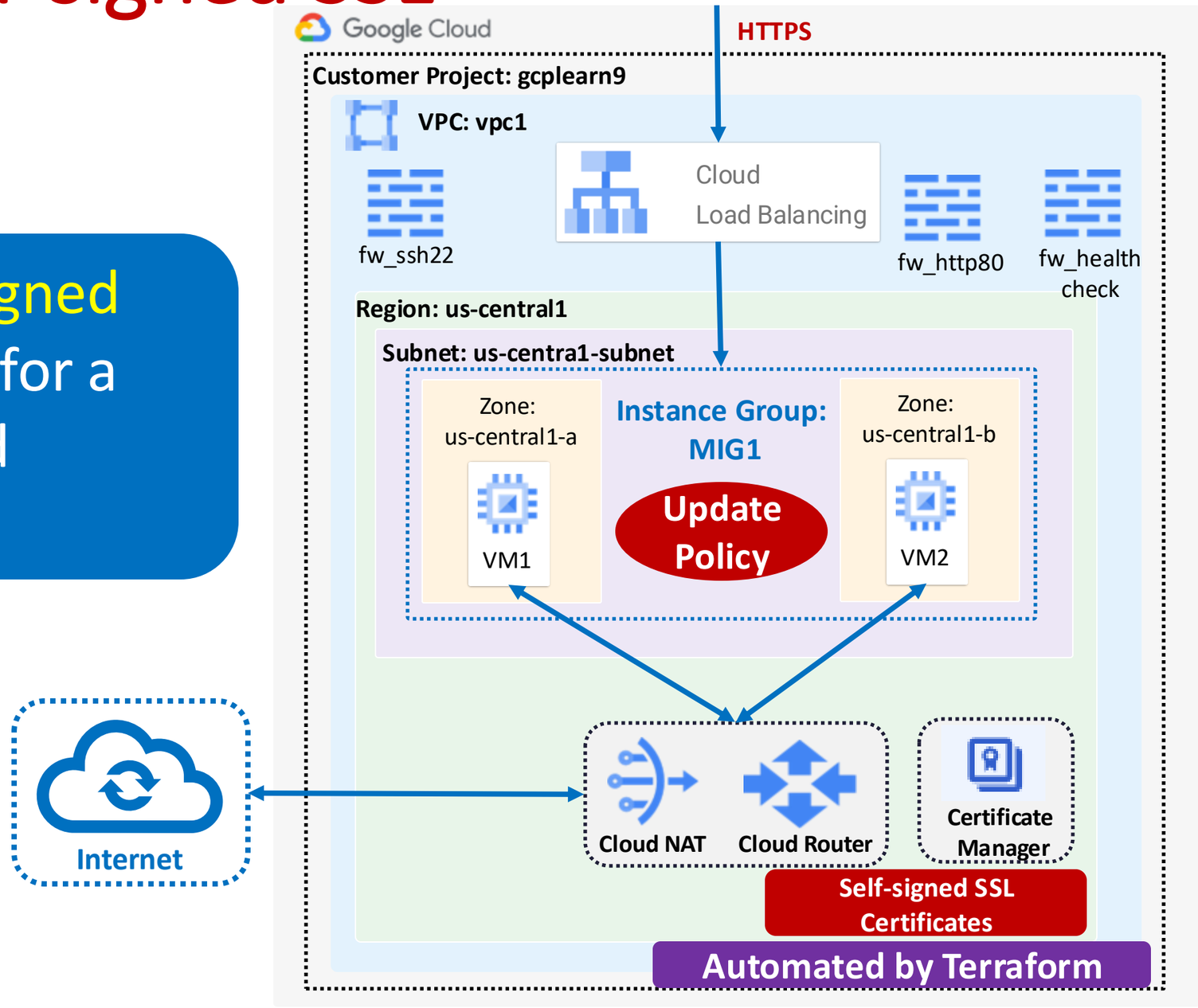
Users

HTTP -> HTTPS Redirect

STACKSIMPLIFY

Use case

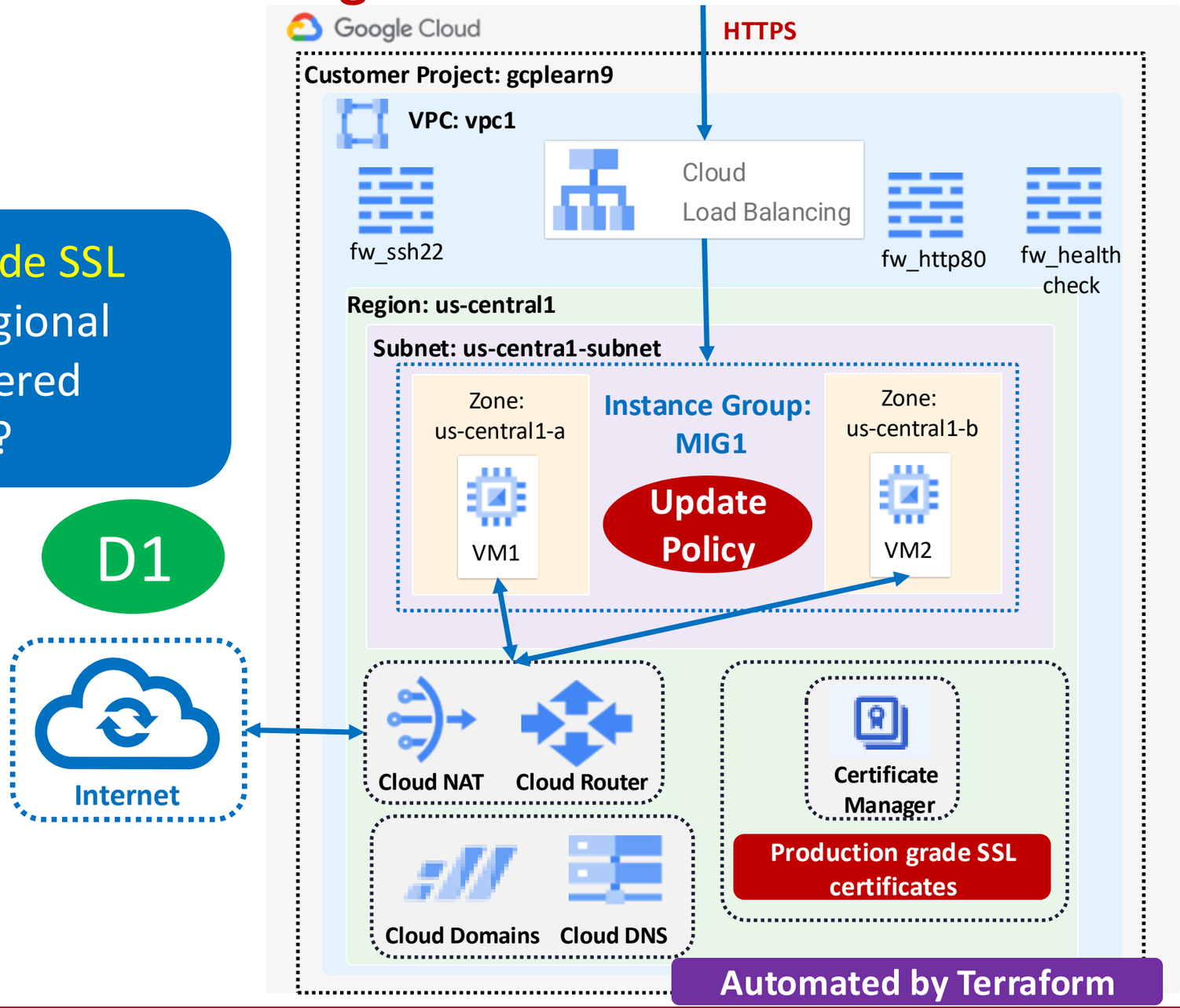
How do you implement self-signed SSL using Certificate Manager for a GCP Regional Application Load Balancer?



GCP Cloud DNS + Cloud Domains as Domain Registrar

Use case

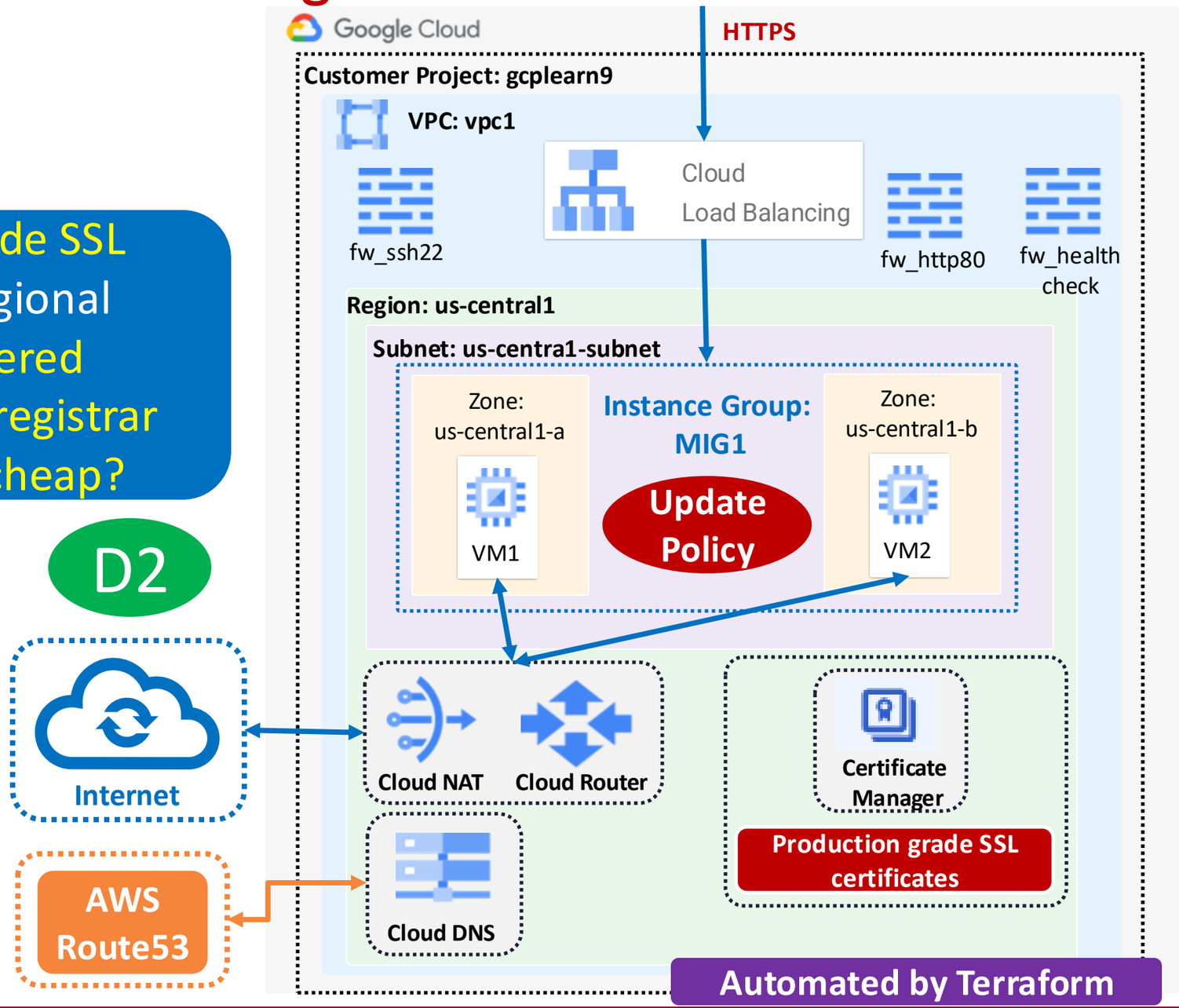
How do you implement production-grade SSL using Certificate Manager for a GCP Regional Application Load Balancer with a registered domain present in GCP Cloud Domains?



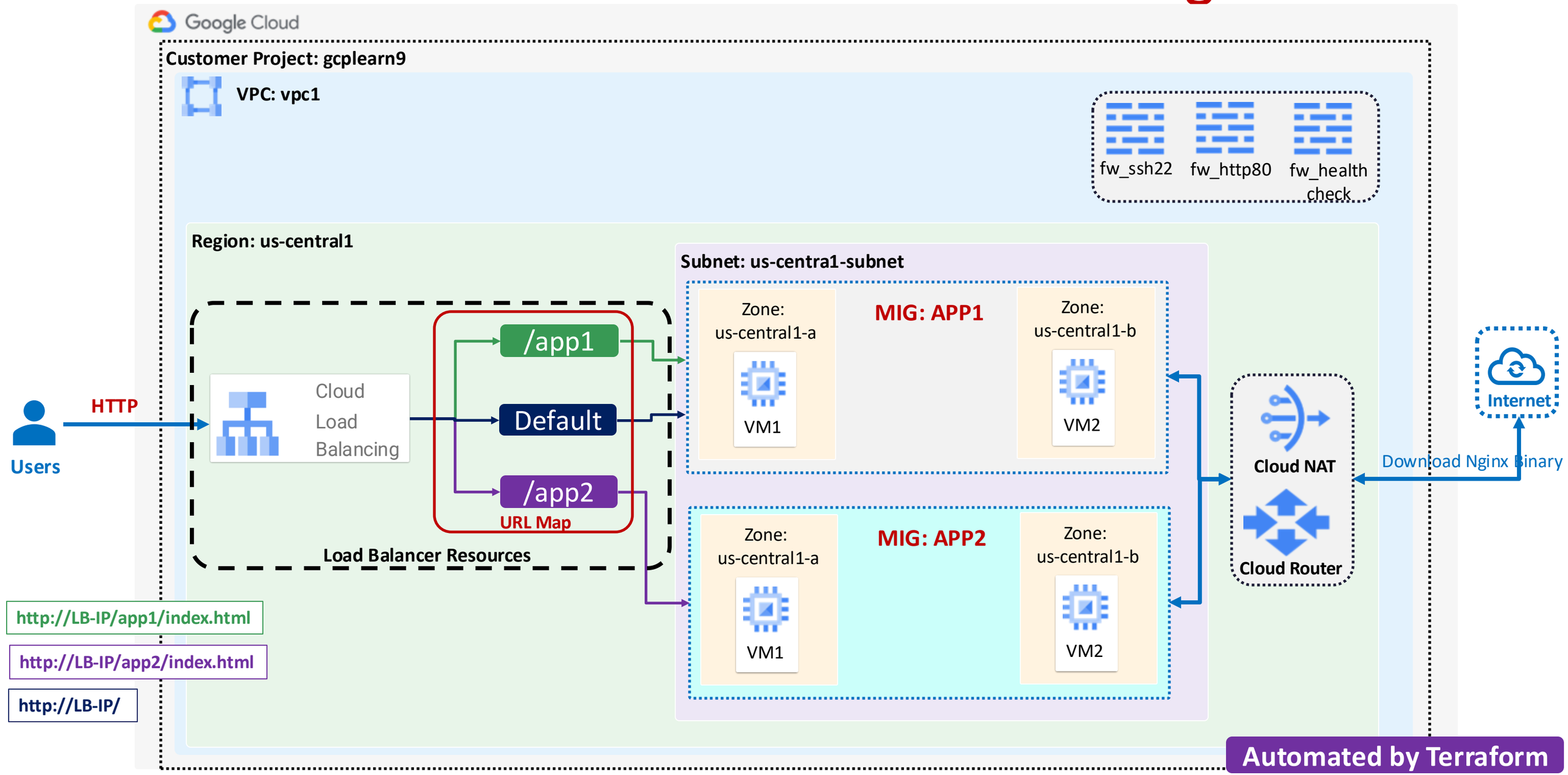
GCP Cloud DNS + AWS Route53 as Domain Registrar

Use case

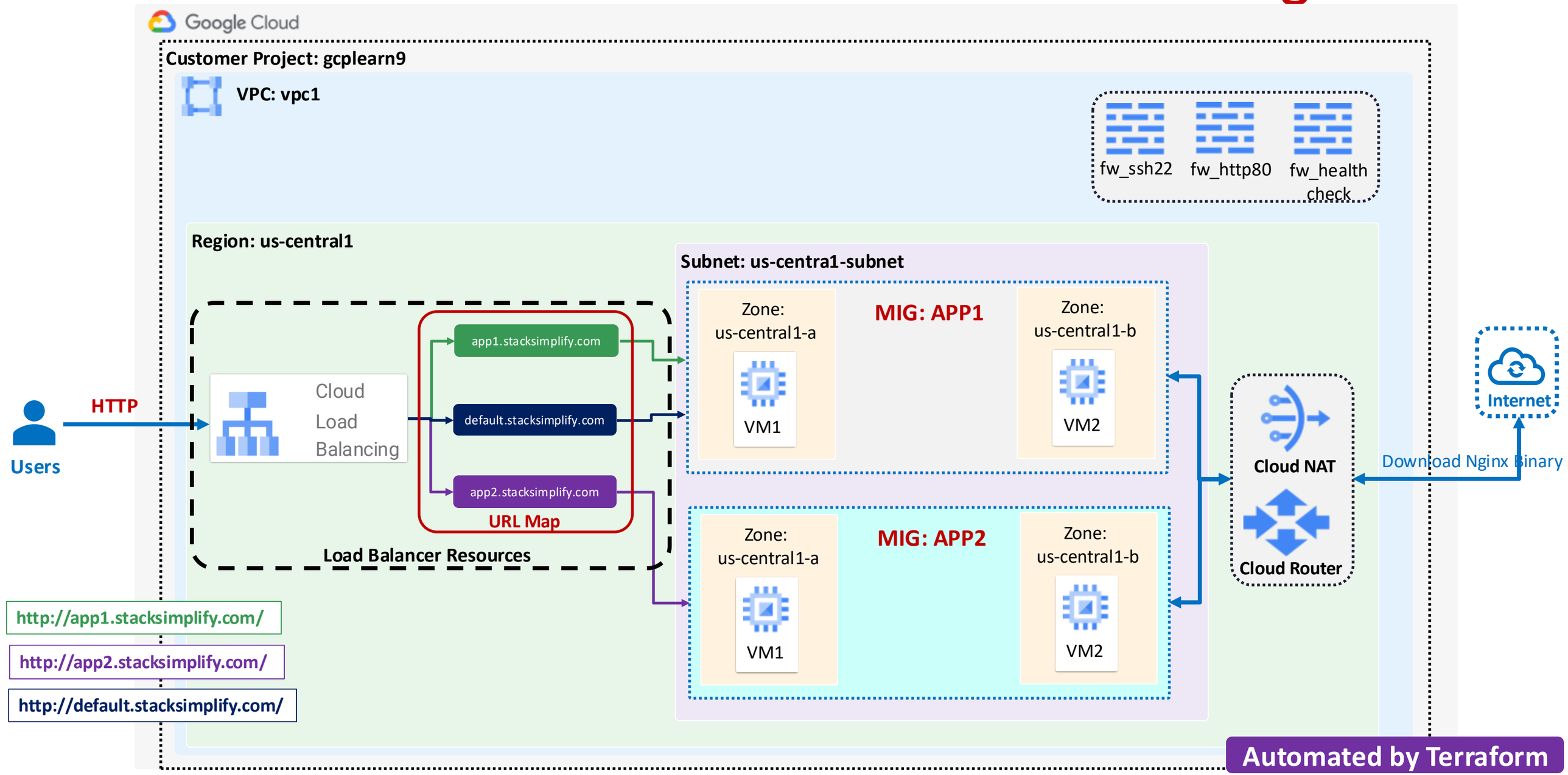
How do you implement production-grade SSL using Certificate Manager for a GCP Regional Application Load Balancer with a registered domain present in an external domain registrar like AWS Route 53, GoDaddy, or Namecheap?



GCP Load Balancer + Context Path Routing

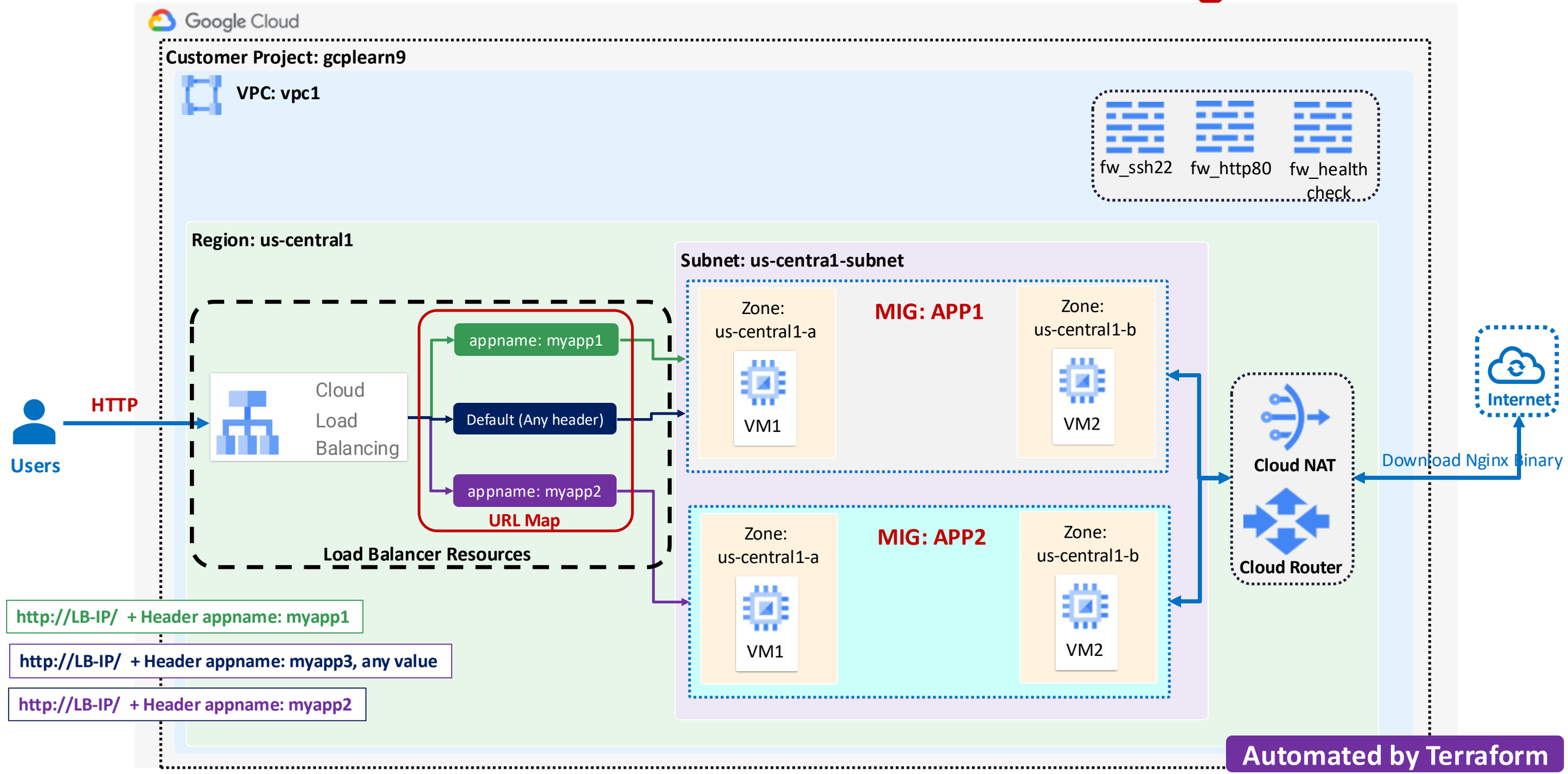


GCP Load Balancer + Domain Name based Routing

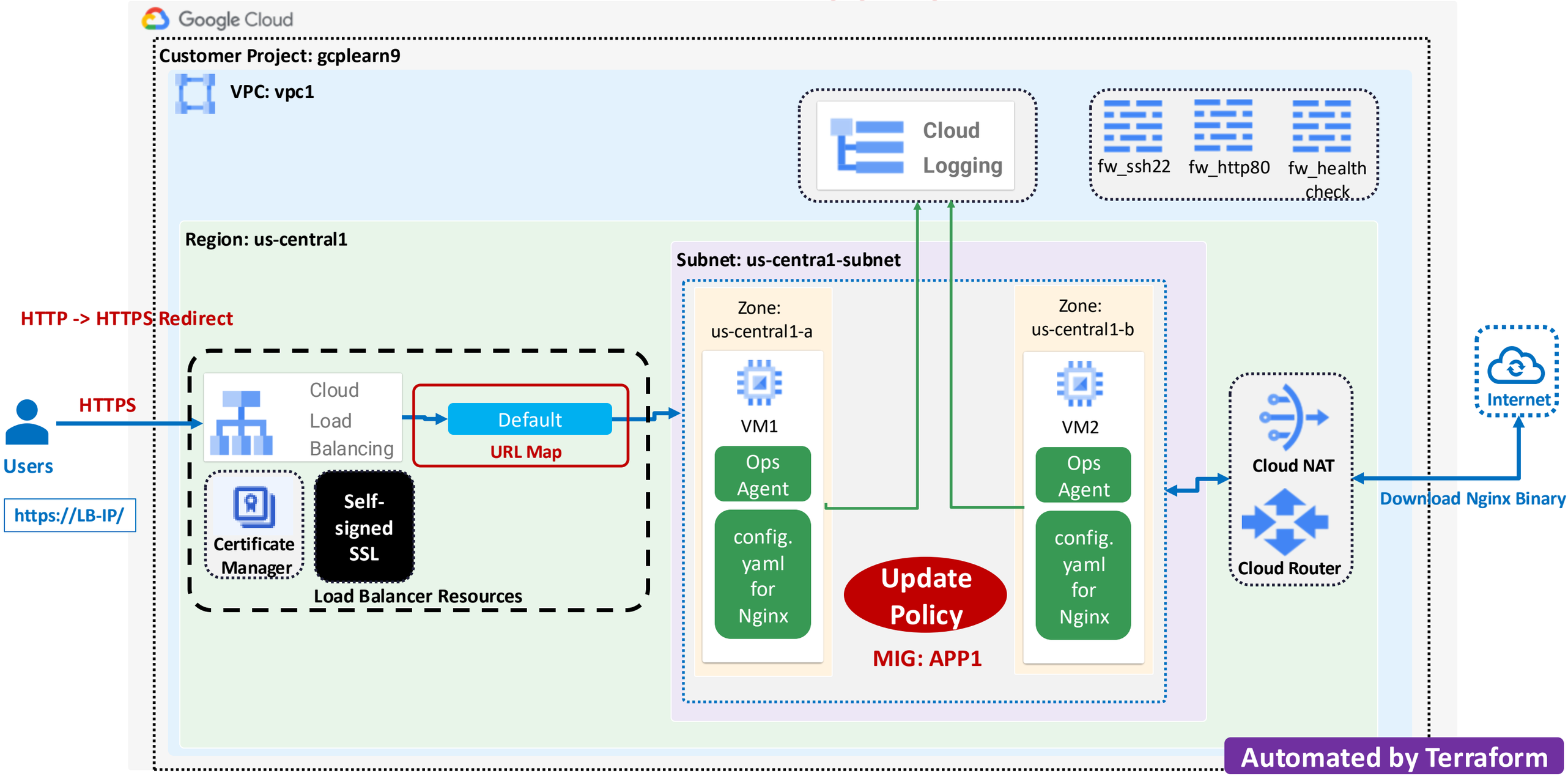


GCP Load Balancer + Header based Routing

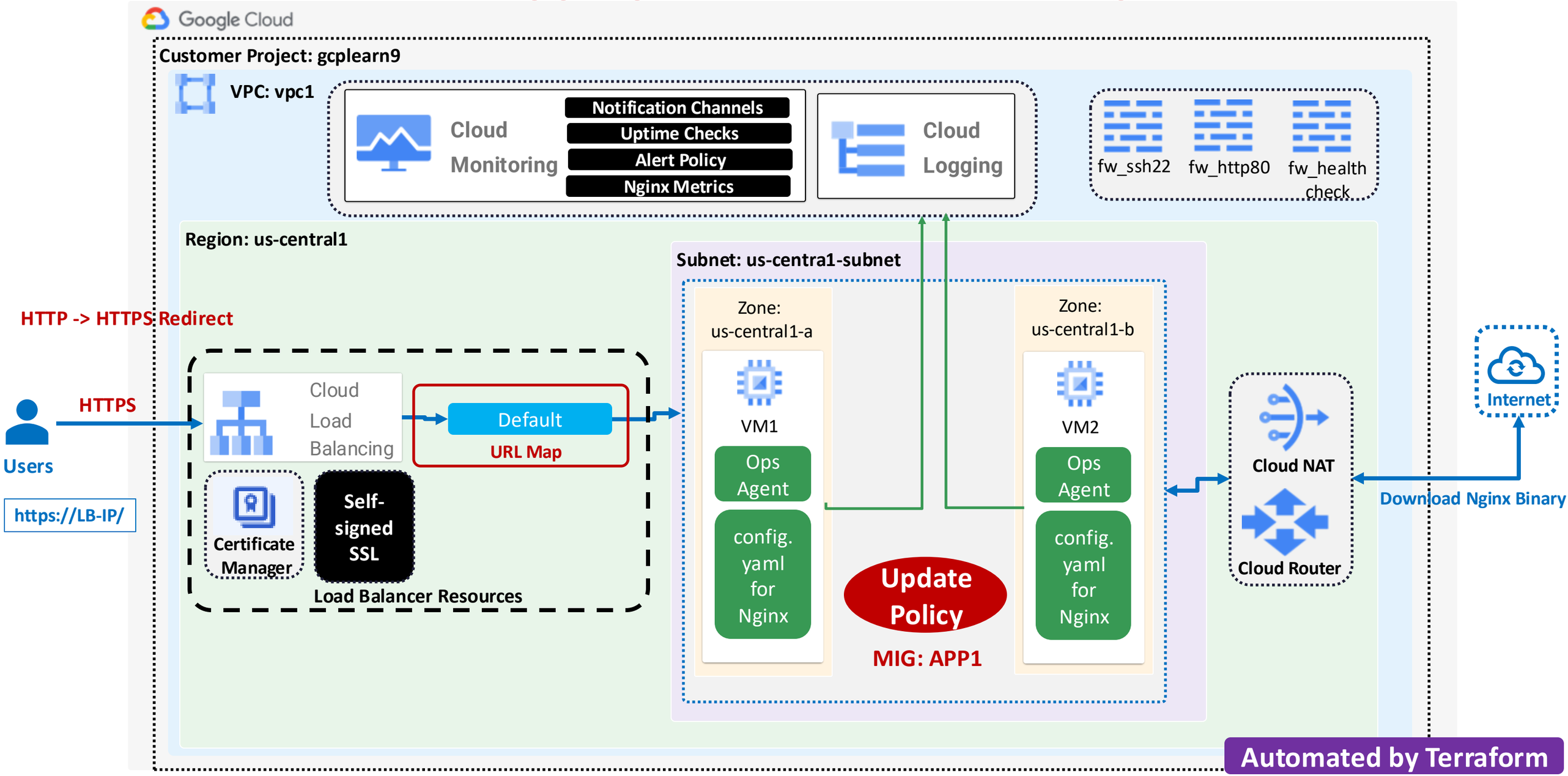
STACKSIMPLIFY



GCP Cloud Logging

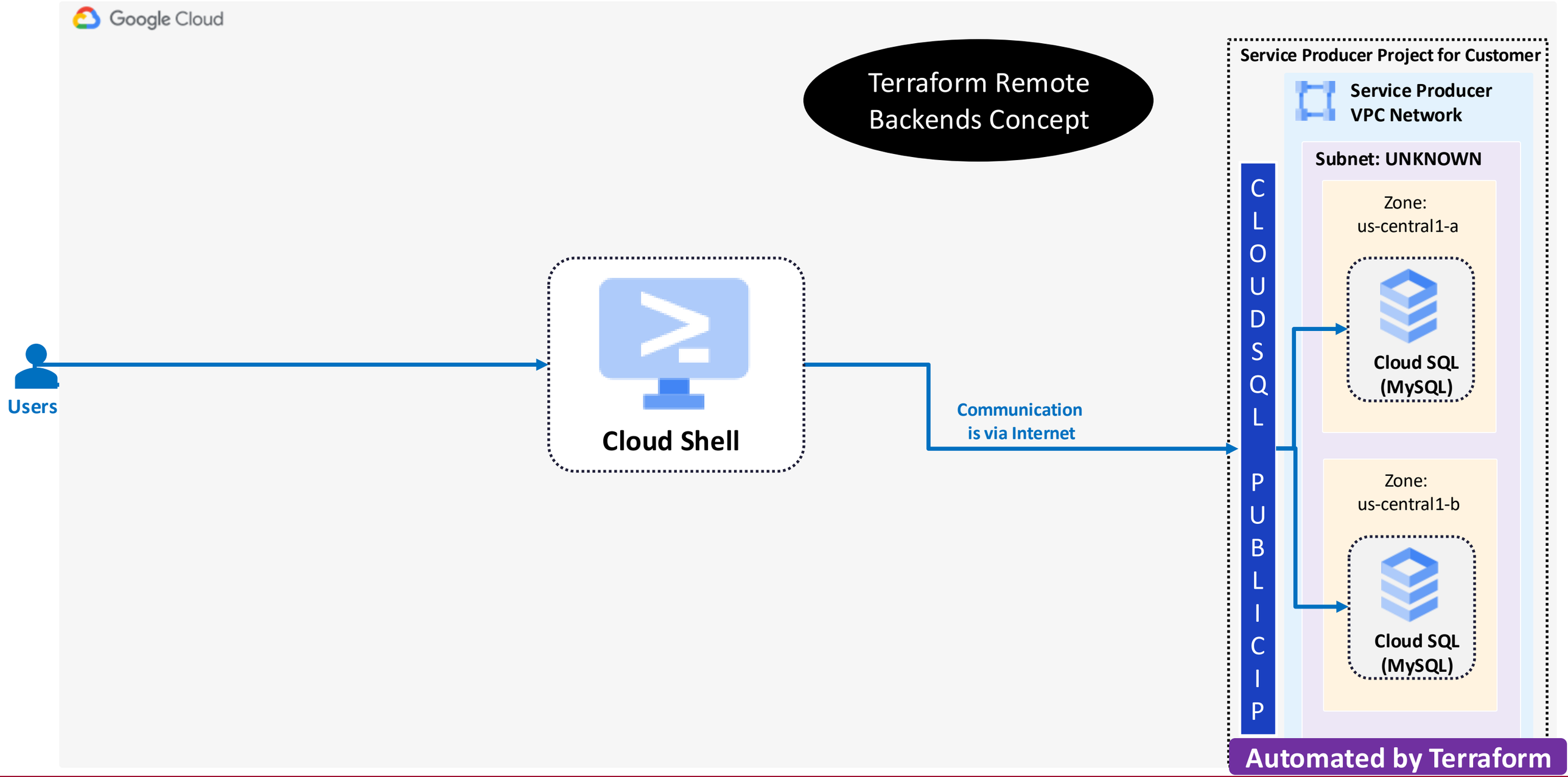


Cloud Logging + Cloud Monitoring



GCP Cloud SQL Public IP

STACKSIMPLIFY



Terraform Remote State Datasource

STACKSIMPLIFY

The terraform_remote_state data source retrieves the root module output values from project-1 Terraform configuration, using the latest state snapshot from the remote backend.

Project-1

Terraform Resources
1. Cloud SQL Database

cloudsql/publicdb/default.tfstate

terraform_remote_state
data source

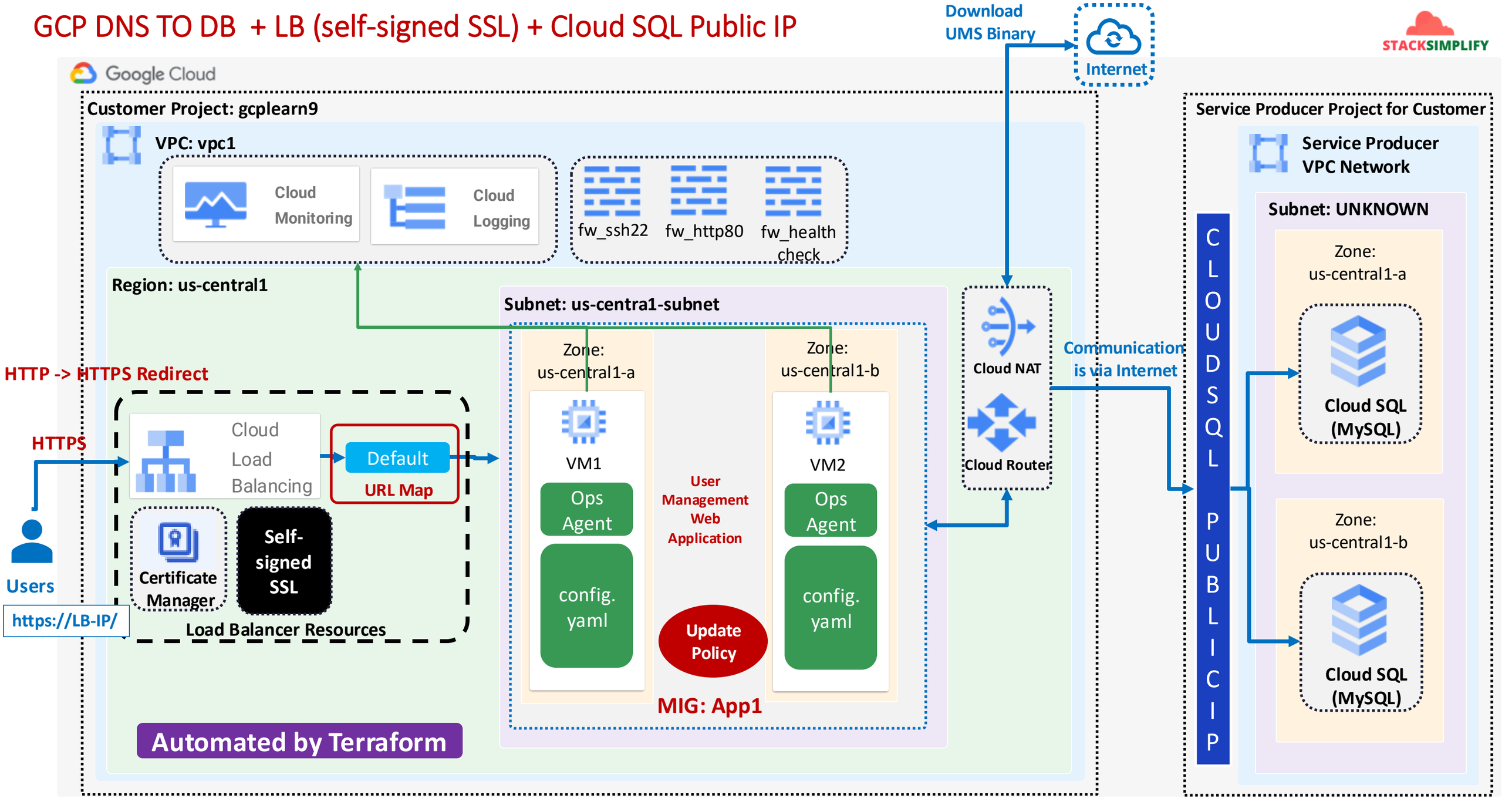
Outputs from Project-1
1. Cloud SQL Public IP

Project-2

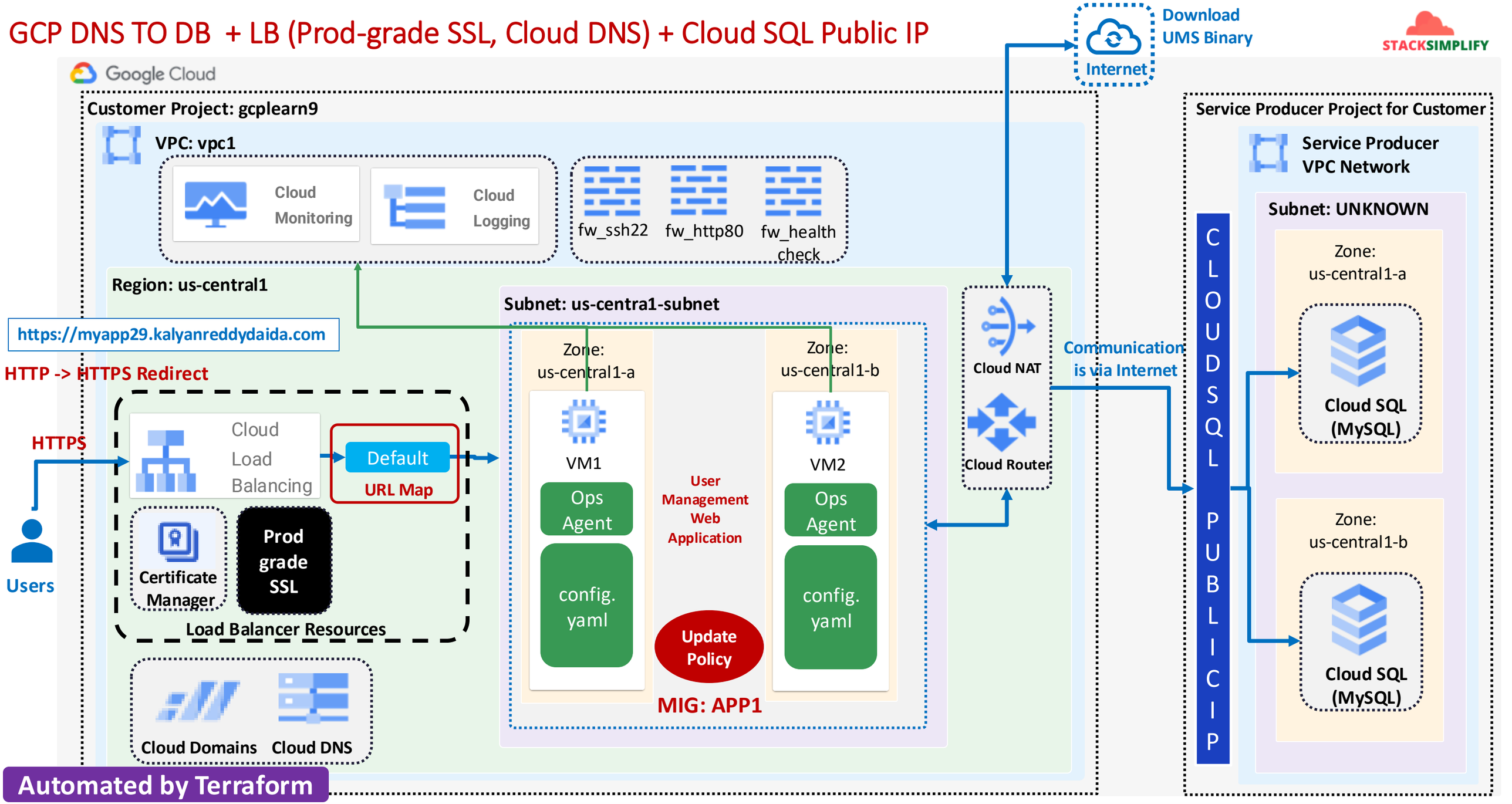
Terraform Resources
1. **VPC, Subnets**
2. **Instance Templates, MIG**
3. **Service Account**
4. **Load Balancer**
5. **Cloud NAT, Cloud Router**
6. **Certificate Manager**
7. **Uptime Checks**

myapp1/httpslb-selfsigned-publicdb/default.tfstate

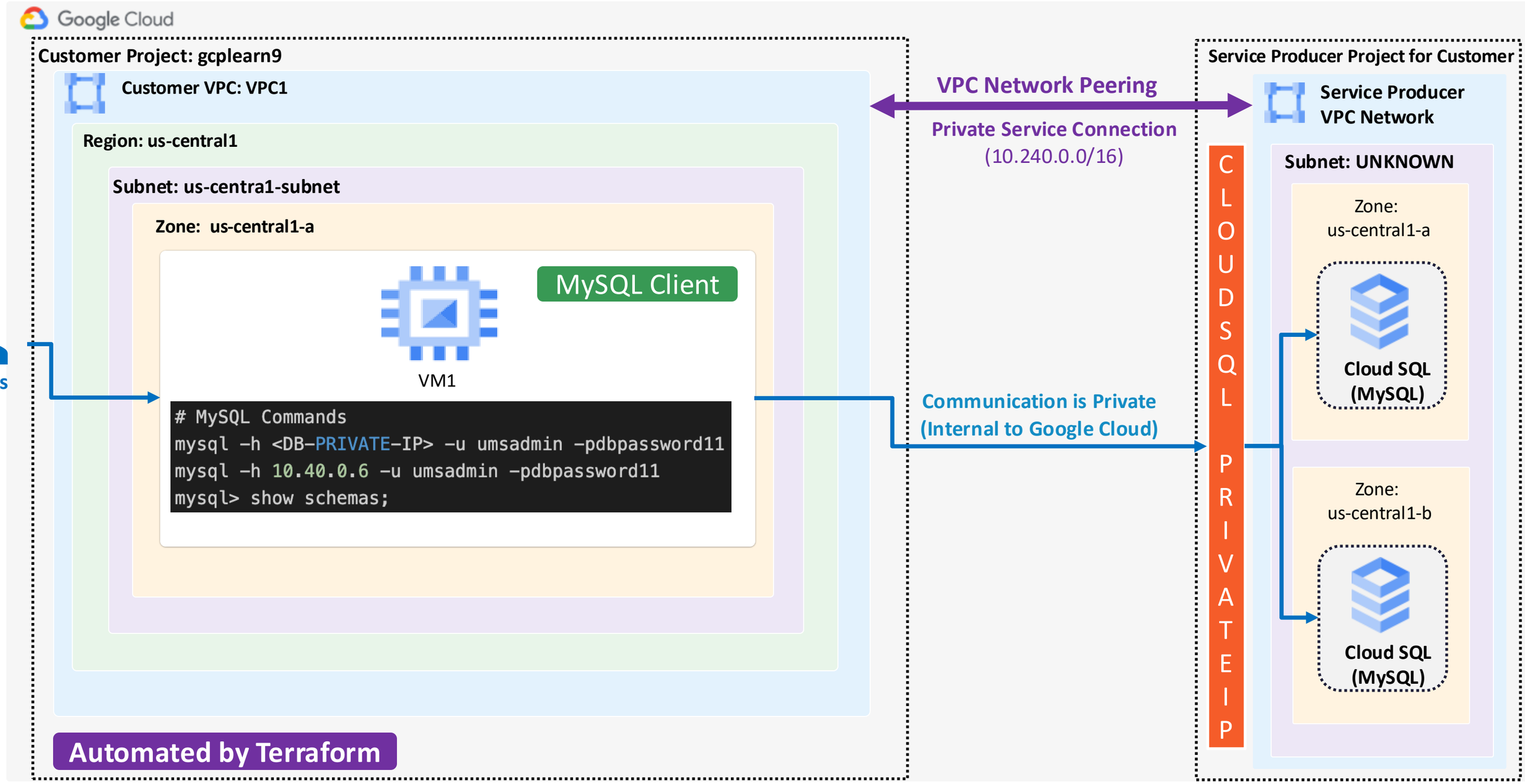
GCP DNS TO DB + LB (self-signed SSL) + Cloud SQL Public IP



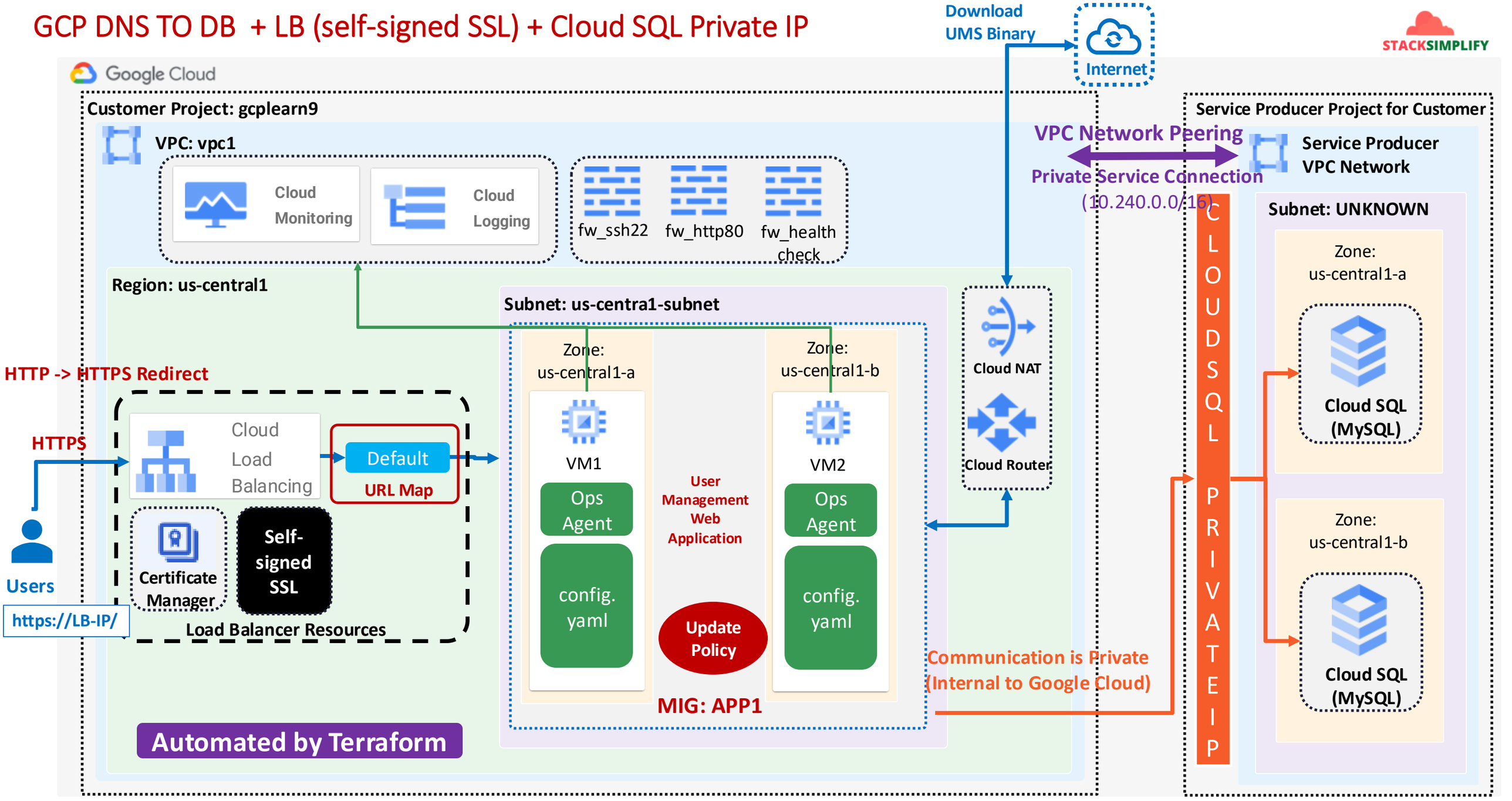
GCP DNS TO DB + LB (Prod-grade SSL, Cloud DNS) + Cloud SQL Public IP



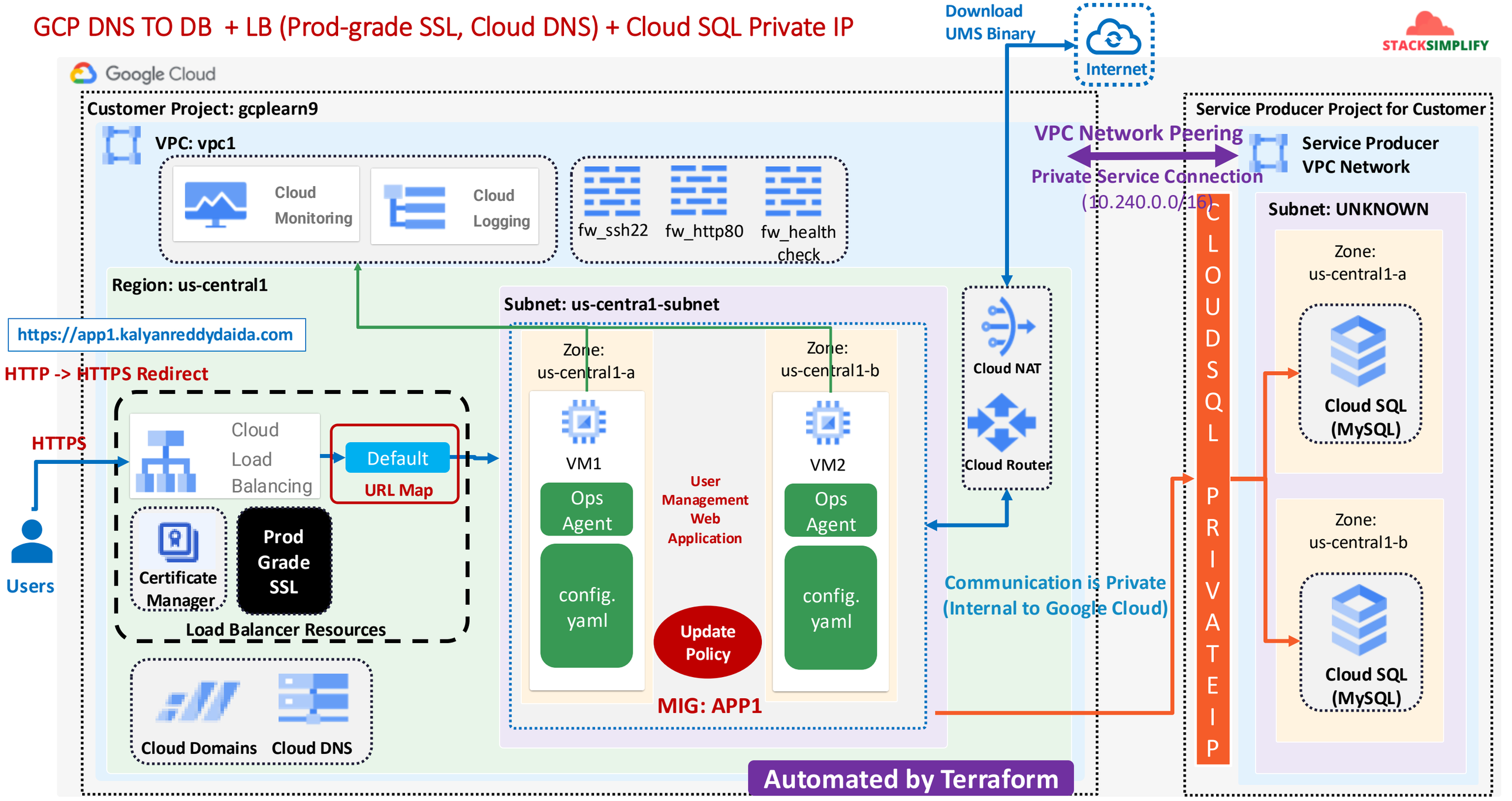
GCP Cloud SQL Private IP



GCP DNS TO DB + LB (self-signed SSL) + Cloud SQL Private IP



GCP DNS TO DB + LB (Prod-grade SSL, Cloud DNS) + Cloud SQL Private IP



Terraform Registry - Use Publicly Available Modules

The Terraform Registry hosts a broad collection of publicly available Terraform modules for configuring many kinds of common infrastructure.

Modules Demo 1

These modules are free to use, and Terraform can download them automatically if you specify the appropriate source and version in a module call block.

```
# Module: VPC
module "vpc" {
    source  = "terraform-google-modules/network/google"
    version = "~> 9.1"
    project_id   = var.gcp_project
    network_name = "${local.name}-vpc"
    routing_mode = "GLOBAL"
    subnets = [
        {
            subnet_name           = "${local.name}-${var.gcp_region1}-subnet"
            subnet_ip             = "10.128.0.0/20"
            subnet_region         = var.gcp_region1
        }
    ]
}
```

Build a Local Terraform Module

STACKSIMPLIFY

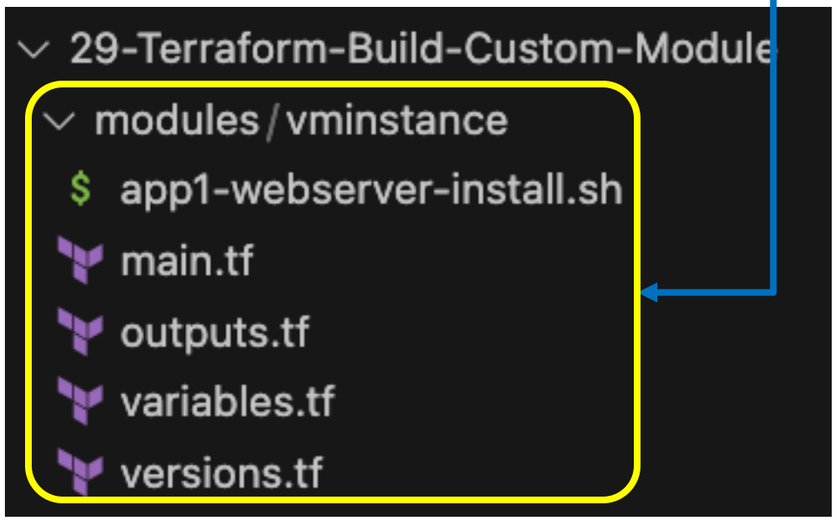
Modules
Demo 2

Build a Local Terraform Module and call it from Root Module and Test

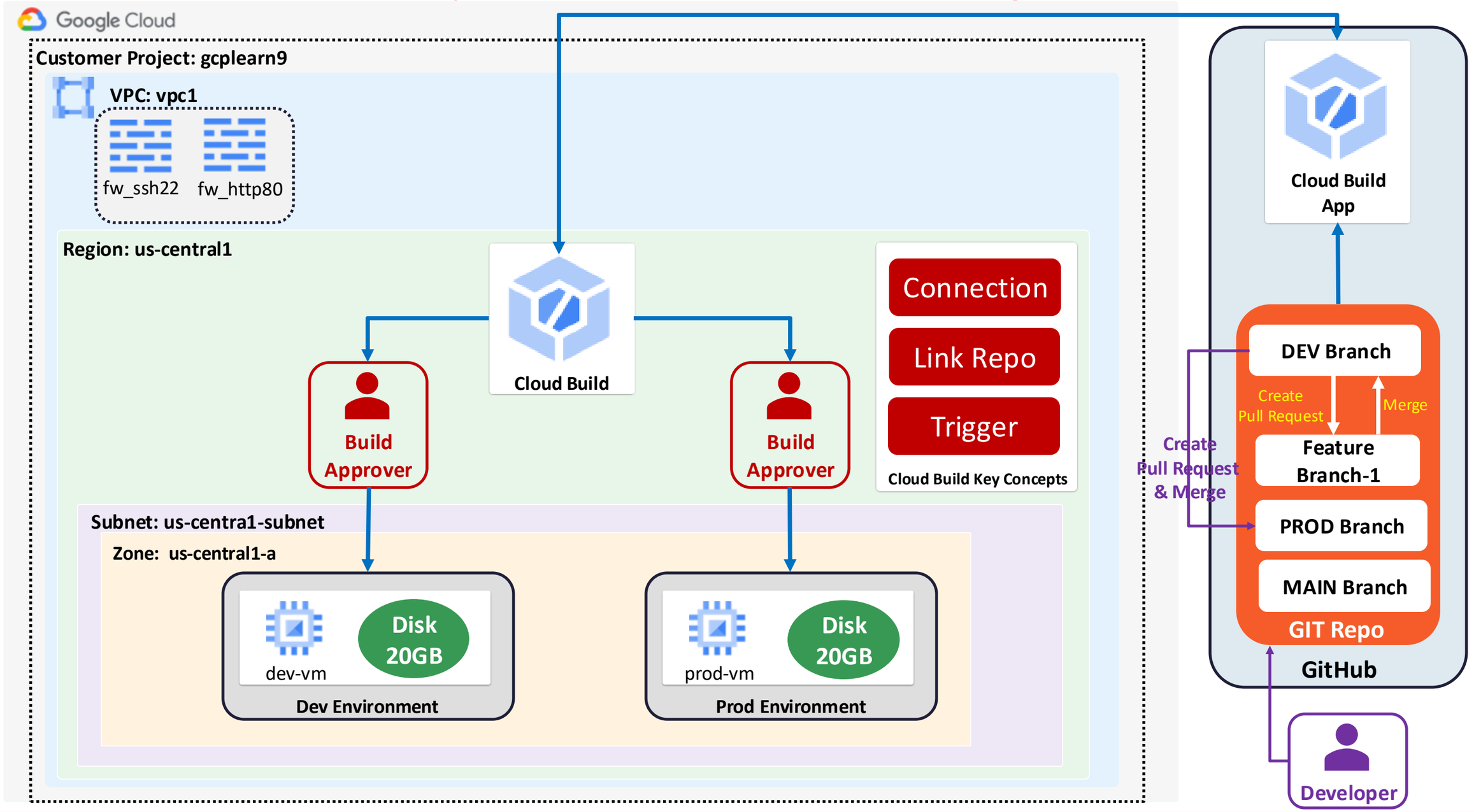
Root Module

Call VM Instance child module in Root Module

Child Module

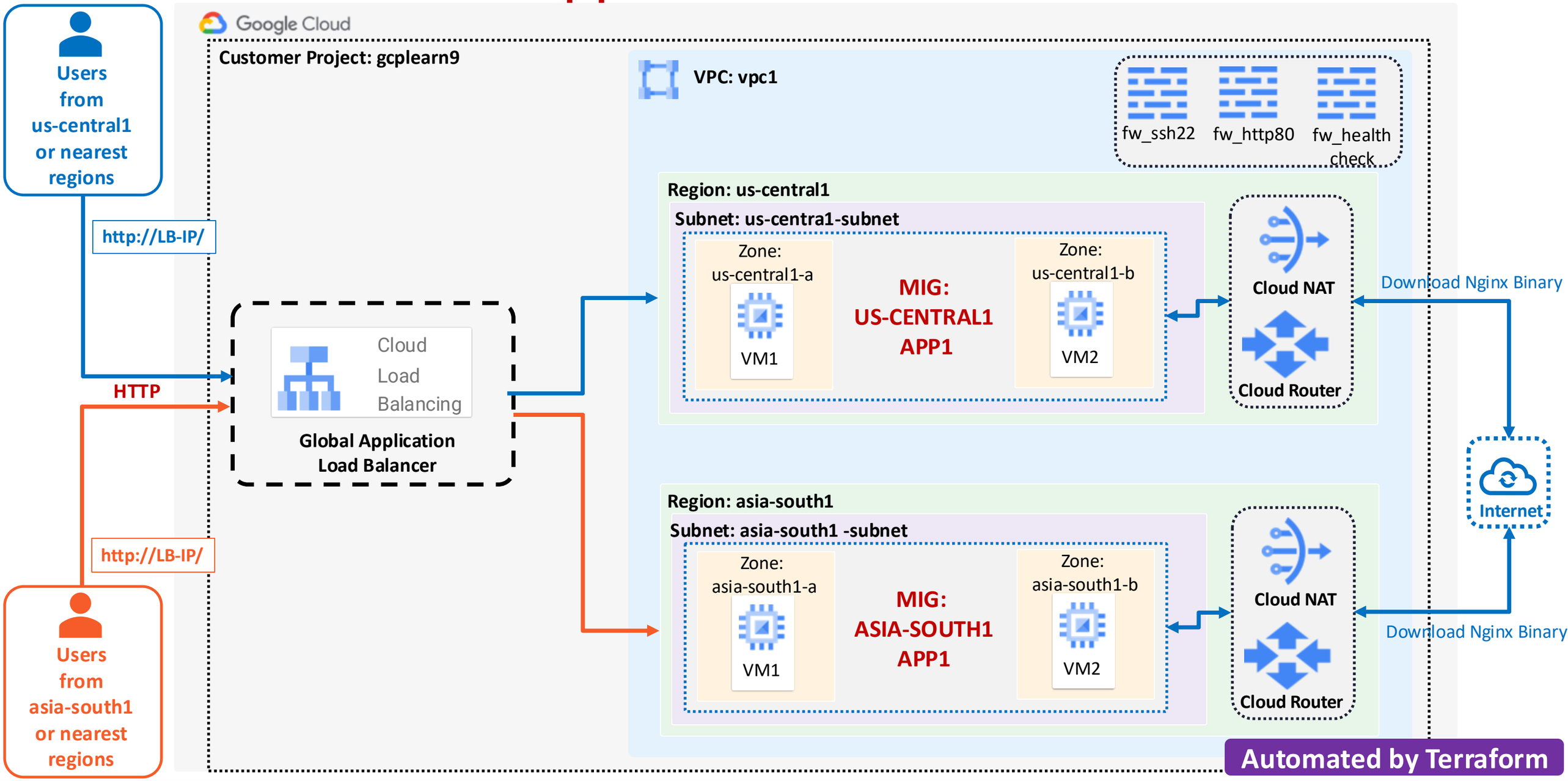


GCP DevOps for Terraform Configs



GCP Global Application Load Balancer

STACKSIMPLIFY



STACKSIMPLIFY

END OF INTRODUCTION

Demo

Terraform Installation CLI Tools

Terraform Installation

STACKSIMPLIFY

gcloud CLI

Terraform CLI

VS Code Editor

Terraform plugin for VS Code

Mac OS

Windows OS

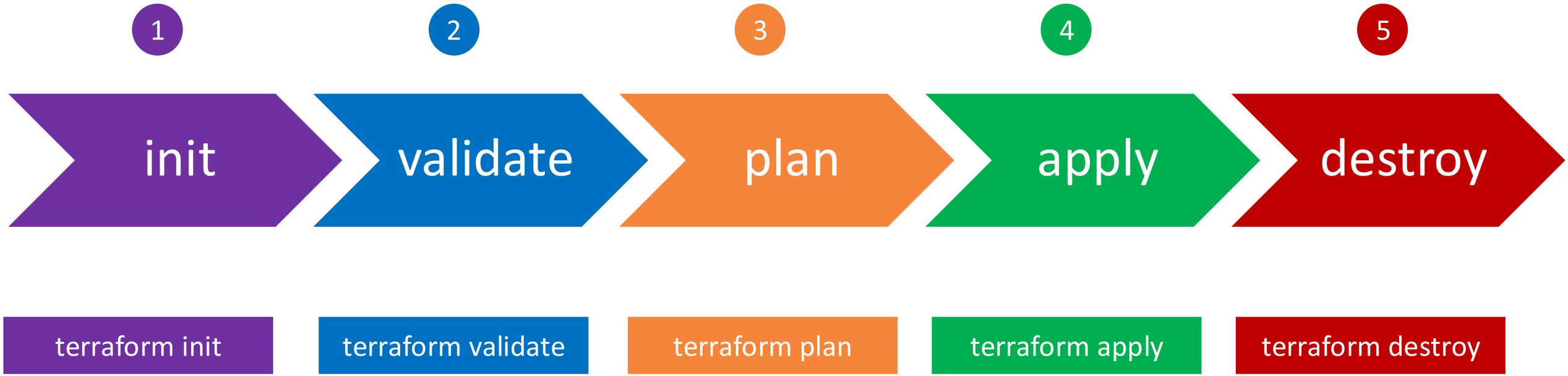
Demo

Terraform

Command Basics

STACKSIMPLIFY

Terraform Workflow



Terraform Workflow

1 **init**

- Used to Initialize a working directory containing terraform config files
- This is the first command that should be run after writing a new Terraform configuration
- Downloads Providers

2 **validate**

- Validates the terraform configurations files in that respective directory to ensure they are syntactically valid and internally consistent.

3 **plan**

- Creates an execution plan
- Terraform performs a refresh and determines what actions are necessary to achieve the desired state specified in configuration files

4 **apply**

- Used to apply the changes required to reach the desired state of the configuration.
- By default, apply scan s the current directory for the configuration and applies the changes appropriately.

5 **destroy**

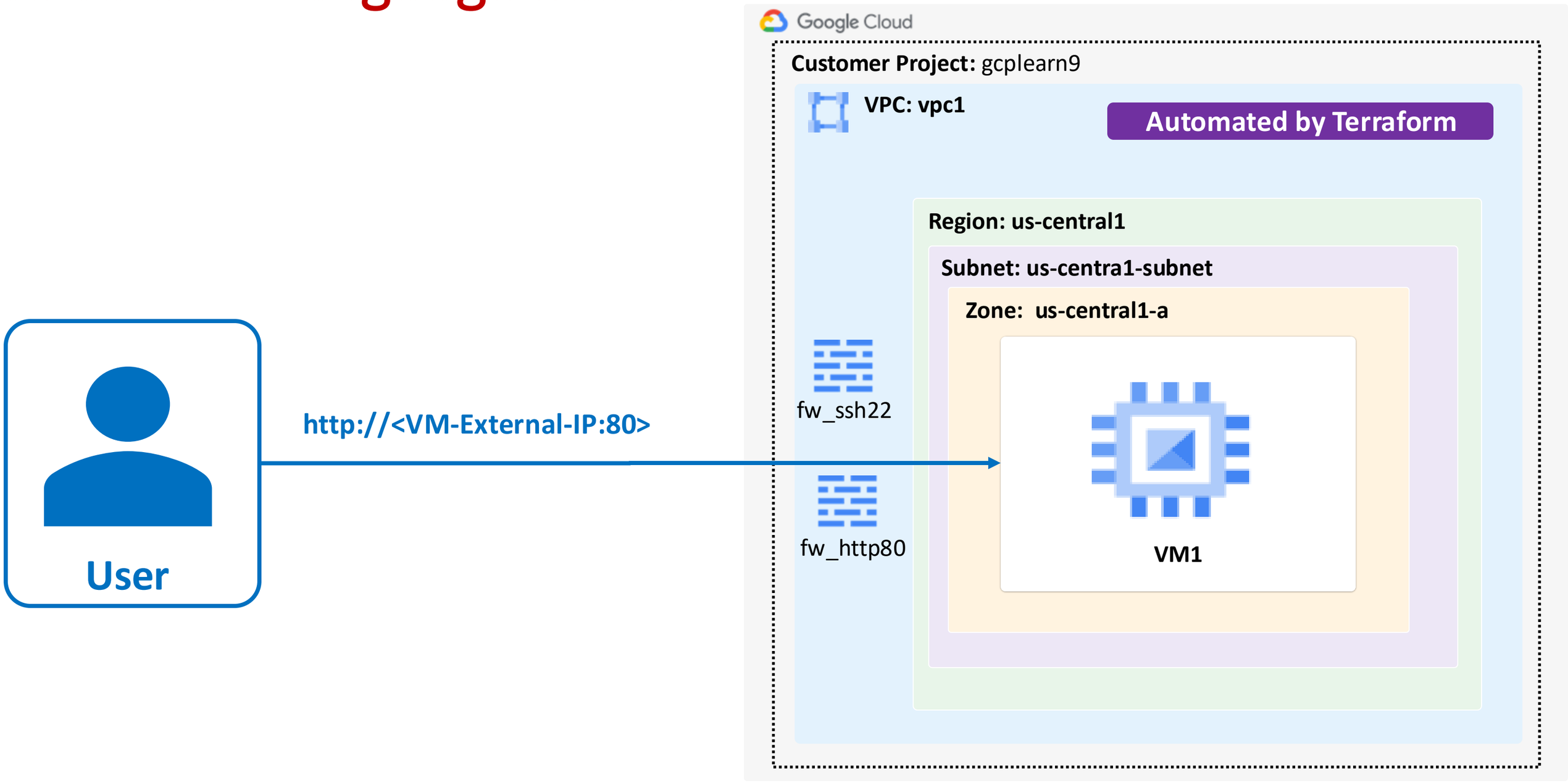
- Used to destroy the Terraform-managed infrastructure
- This will ask for confirmation before destroying.

Demo

Terraform

What are we going to automate as part of this demo using Terraform?

Terraform Language Basics



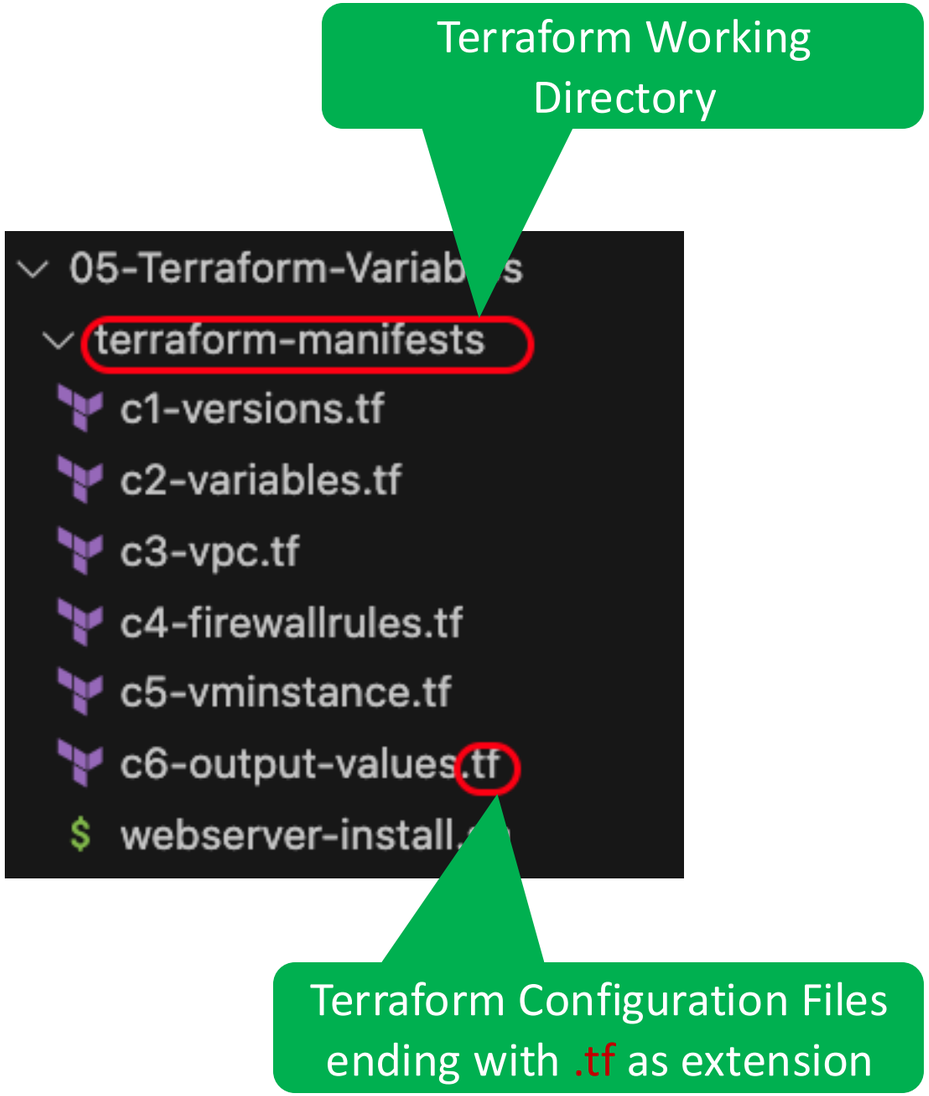
Demo

Terraform

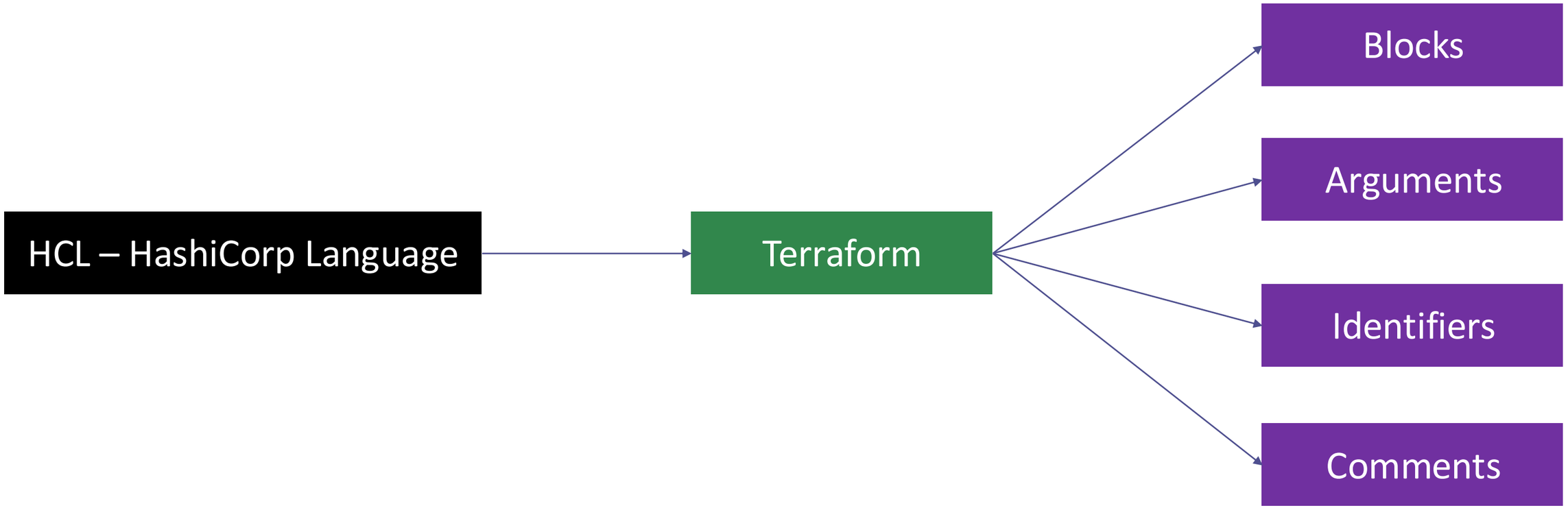
Language Basics

Terraform Language Basics - Files

- Code in the Terraform language is stored in plain text files with the .tf file extension.
- There is also a JSON-based variant of the language that is named with the .tf.json file extension.
- We can call the files containing terraform code as Terraform Configuration Files or Terraform Manifests
- Terraform files should be saved in a directory (Terraform Working Directory)



Terraform Language Basics - Configuration Syntax



Terraform Language Basics - Configuration Syntax

STACKSIMPLIFY

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block Body
  <IDENTIFIER> = <EXPRESSION> # Argument
}

# Resource: Subnet
resource "google_compute_subnetwork" "mysubnet" {
  name = "subnet1"
  region = "us-central1"
  ip_cidr_range = "10.128.0.0/20"
  network = google_compute_network.myvpc.id
}
```

Based on Block Type, block labels can be 1 or 2
Example:
Resources: 2 block labels
Variables: 1 block label

Block Labels

Block Type

Arguments

Top Level & Block inside Blocks

Top Level Blocks: resource, provider, terraform, variable, data, module, output

Block Inside Block: provisioners, resource specific blocks like tags

Terraform Language Basics - Configuration Syntax

STACKSIMPLIFY

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block Body
  <IDENTIFIER> = <EXPRESSION> # Argument
}

# Resource: Subnet
resource "google_compute_subnetwork" "mysubnet" {
  name          = "subnet1"
  region        = "us-central1"
  ip_cidr_range = "10.128.0.0/20"
  network       = google_compute_network.myvpc.id
}
```

Argument Name [OR] Identifier

Argument Value [OR] Expression

Terraform Language Basics - Configuration Syntax

STACKSIMPLIFY

Multi-line comment /* ..... */

Single Line Comments with # or //

```
/*
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {
  # Block Body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
*/

# Resource: Subnet
resource "google_compute_subnetwork" "mysubnet" {
  name          = "subnet1"
  region        = "us-central1"
  ip_cidr_range = "10.128.0.0/20"
  network       = google_compute_network.myvpc.id // GET VPC ID
}
```

Terraform language uses a limited number of **top-level block** types, which are **blocks** that can appear outside of any other **block** in a TF configuration file.

Terraform Top-Level Blocks

Most of **Terraform's features** are implemented as **top-level** blocks.

- Terraform Block
- Providers Block
- Resources Block

Fundamental Blocks

- Input Variables Block
- Output Values Block
- Local Values Block

Variable Blocks

- Data Sources Block
- Modules Block

Calling / Referencing Blocks

- Import Block
- Moved Block
- Removed Block
- Check Block

NEW BLOCKS – Added Recently

Demo

Terraform
Fundamental Blocks

Terraform Basic Blocks

STACKSIMPLIFY

Terraform Block

- Special block used to configure some behaviors
- Specifying a required Terraform Version
- Specifying Provider Requirements
- Configuring a Terraform Backend (Terraform State)

Provider Block

- HEART of Terraform
- Terraform relies on providers to interact with Remote Systems
- Declare providers for Terraform to install providers & use them
- Provider configurations belong to Root Module

Resource Block

- Each Resource Block describes one or more Infrastructure Objects
- Resource Syntax: How to declare Resources?
- Resource Behavior: How Terraform handles resource declarations?
- Provisioners: We can configure Resource post-creation actions

Demo

Terraform

Terraform Block

Terraform Block

- This block can be called in 3 ways. All means the same.
  - Terraform Block
  - Terraform Settings Block
  - Terraform Configuration Block
- Each terraform block can contain a number of settings related to Terraform's behavior.
- Within a terraform block, only constant values can be used; arguments may not refer to named objects such as resources, input variables, etc, and may not use any of the Terraform language built-in functions.

```
# Terraform Settings Block
terraform {
  required_version = ">= 1.8"
  required_providers {
    google = {
      source = "hashicorp/google"
      version = ">= 5.26.0"
    }
  }
 backend "gcs" {
   bucket  = "kalyanbucket201"
   prefix  = "terraform/state"
 }
}
```

Demo

Terraform Providers

Terraform Providers

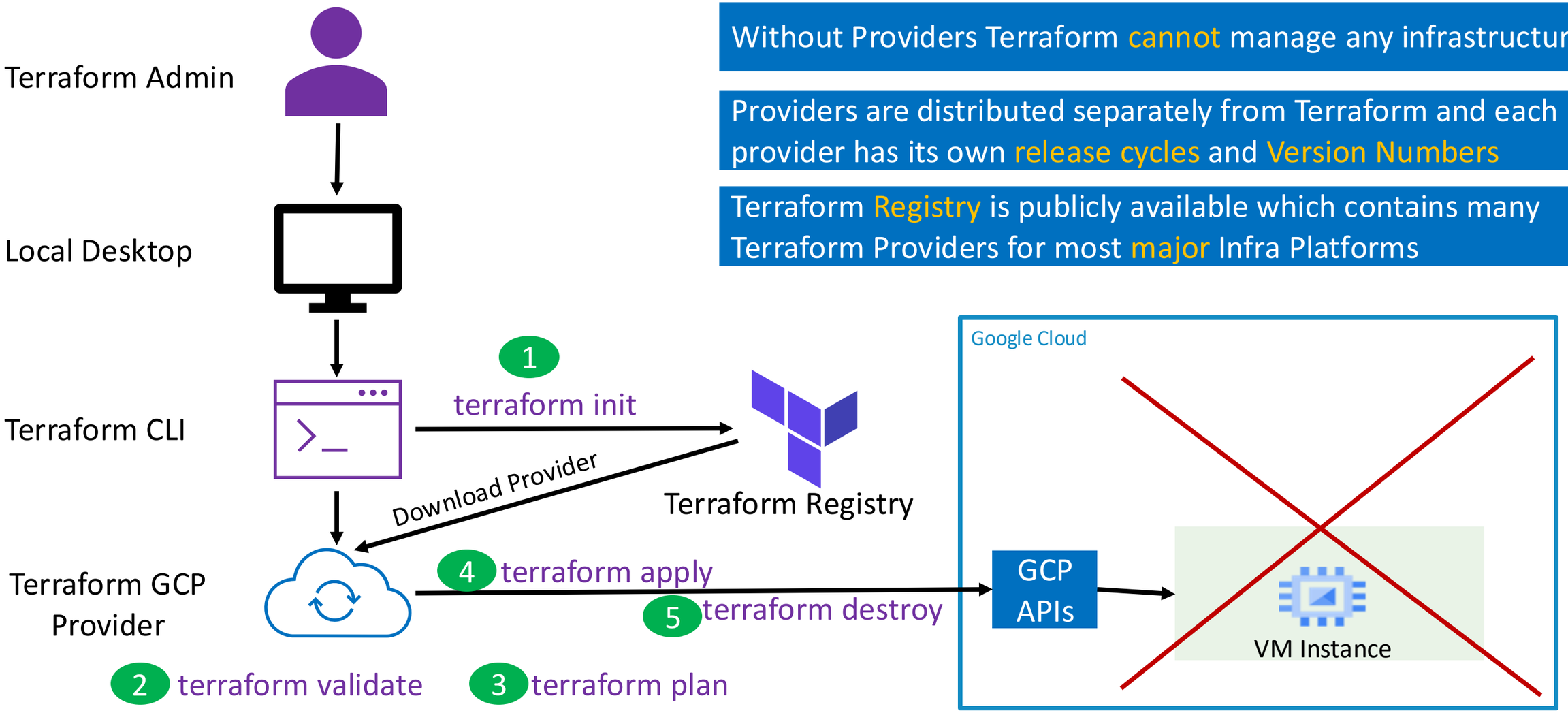
Providers are HEART of Terraform

Every Resource Type (example: VM Instance), is implemented by a Provider

Without Providers Terraform cannot manage any infrastructure.

Providers are distributed separately from Terraform and each provider has its own release cycles and Version Numbers

Terraform Registry is publicly available which contains many Terraform Providers for most major Infra Platforms



Terraform Providers

STACKSIMPLIFY

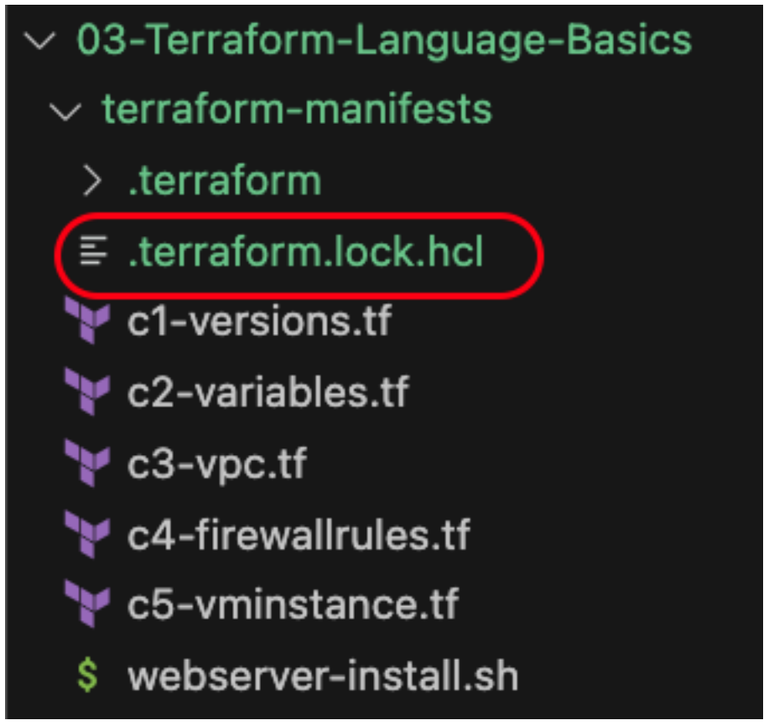
Provider Requirements

```
# Terraform Settings Block
terraform {
  required_version = ">= 1.8"
  required_providers {
    google = {
      source = "hashicorp/google"
      version = ">= 5.26.0"
    }
  }
}
```

Provider Configuration

```
# Terraform Provider Block
provider "google" {
  project = "gcplearn9"
  region = "us-central1"
}
```

Dependency Lock File



Dependency Lock File

STACKSIMPLIFY

terraform-on-google-cloud > 03-Terraform-Language-Basics > terraform-manifests > .terraform.lock.hcl

```
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/google" {
  version     = "5.26.0"
  constraints = ">= 5.26.0"
  hashes = [
    "h1:brgO45d0nWifpX+7NR2CZGtMRnyhapsV7fRRGqkjbkU=",
    "zh:03b9263dcc96b1b27f7c001d95c4b15cb53ab71637520b686490fb9408dde6bc",
    "zh:0fedd2b3bc9e72d5d7f58f663cbfc6cdd4c7ba7d8db5cd86fc3bfb15079ce01a",
    "zh:19339e195365a39e3fc0fc58e107e7002465723cc383ea01eac092e7cf43d081",
    "zh:2e26b07b4c48e161106cdd7649e9b84b8469af16862843b73c073d719e55ab53",
    "zh:326ae5165a65b53b048fd06160e5ac411f0f6038512fb0af754f288c52df501f",
    "zh:3fd2fc68a526e1edf01eae2d8397dbaf18d45592d6d5845c6846c8ad8bb14e68",
    "zh:60d55d0e268b7ca08ef0f883883a65655ef4bdcb7c6af102bce03686c3e1753a",
    "zh:8ea48e74332a57a1a860e66bb2360aaf920f1d71347b48dfe8c8adc1aec7e165",
    "zh:a7457fc8c0bce10891de214456d239997eec6761696dc2d9c1617728d6324ab0",
    "zh:b0a76f59d6aeb10a4ce65696dc4738b43260f2e1161b23285cc05cc5066a42f4",
    "zh:d32aa14ef25eb479f636e3ec4b2cd671cf523c2995cec180e7afe246c903532f",
    "zh:f569b65999264a9416862bca5cd2a6177d94ccb0424f3a4ef424428912b9cb3c",
  ]
}
```

Required Providers

STACKSIMPLIFY

```
# Terraform Settings Block
terraform {
  required_version = ">= 1.8"
  required_providers {
    google = {
      source = "hashicorp/google"
      version = ">= 5.26.0"
    }
  }
}

# Terraform Provider Block
provider "google" {
  project = "gcplearn9"
  region = "us-central1"
}
```

Local Names

Local Names are Module specific and should be unique per-module

Terraform configurations always refer to local name of provider outside required_provider block

Users of a provider can choose any local name for it (mygoogle, google1).

Recommended way of choosing local name is to use preferred local name of that provider (For Google Provider: hashicorp/google, preferred local name is google)

Source

It is the primary location where we can download the Terraform Provider

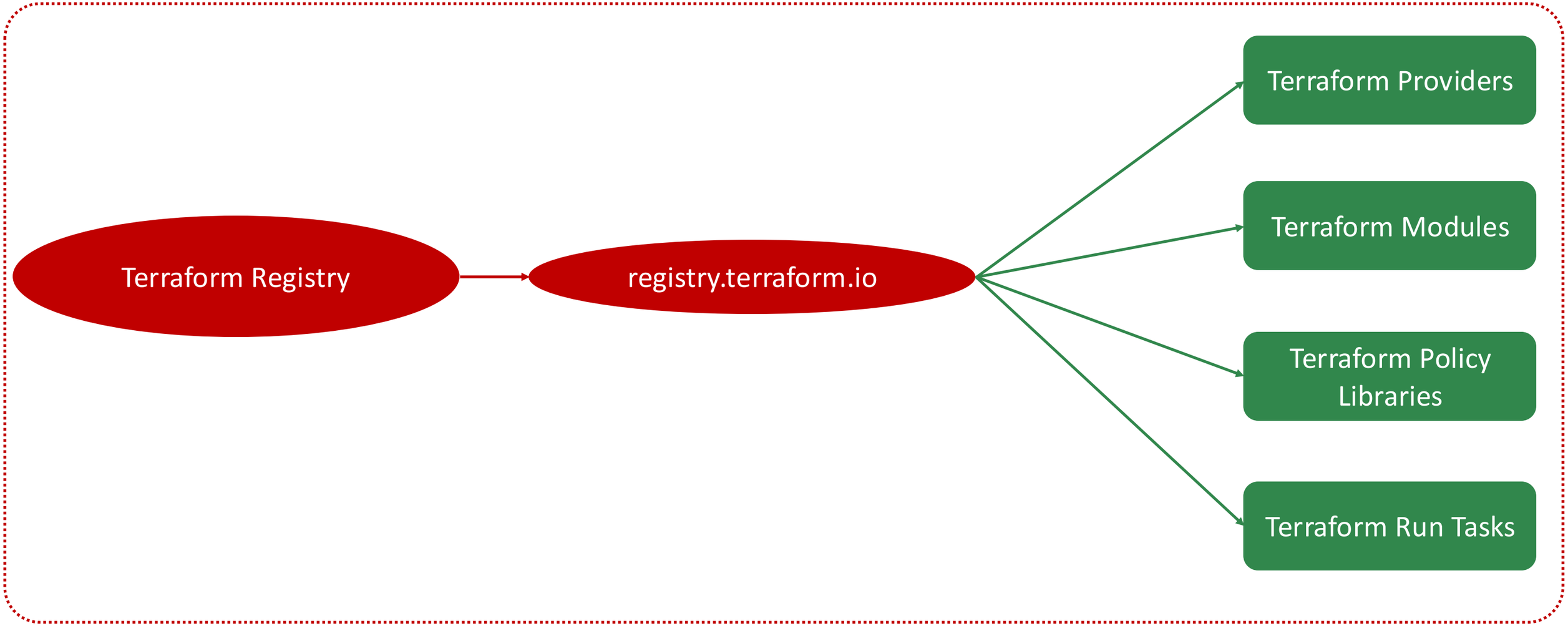
Source addresses consist of three parts delimited by slashes (/)

[<HOSTNAME>/]<NAMESPACE>/<TYPE>

registry.terraform.io/hashicorp/google

Registry Name is optional as default is going to be Terraform Public Registry

Terraform Registry



Demo

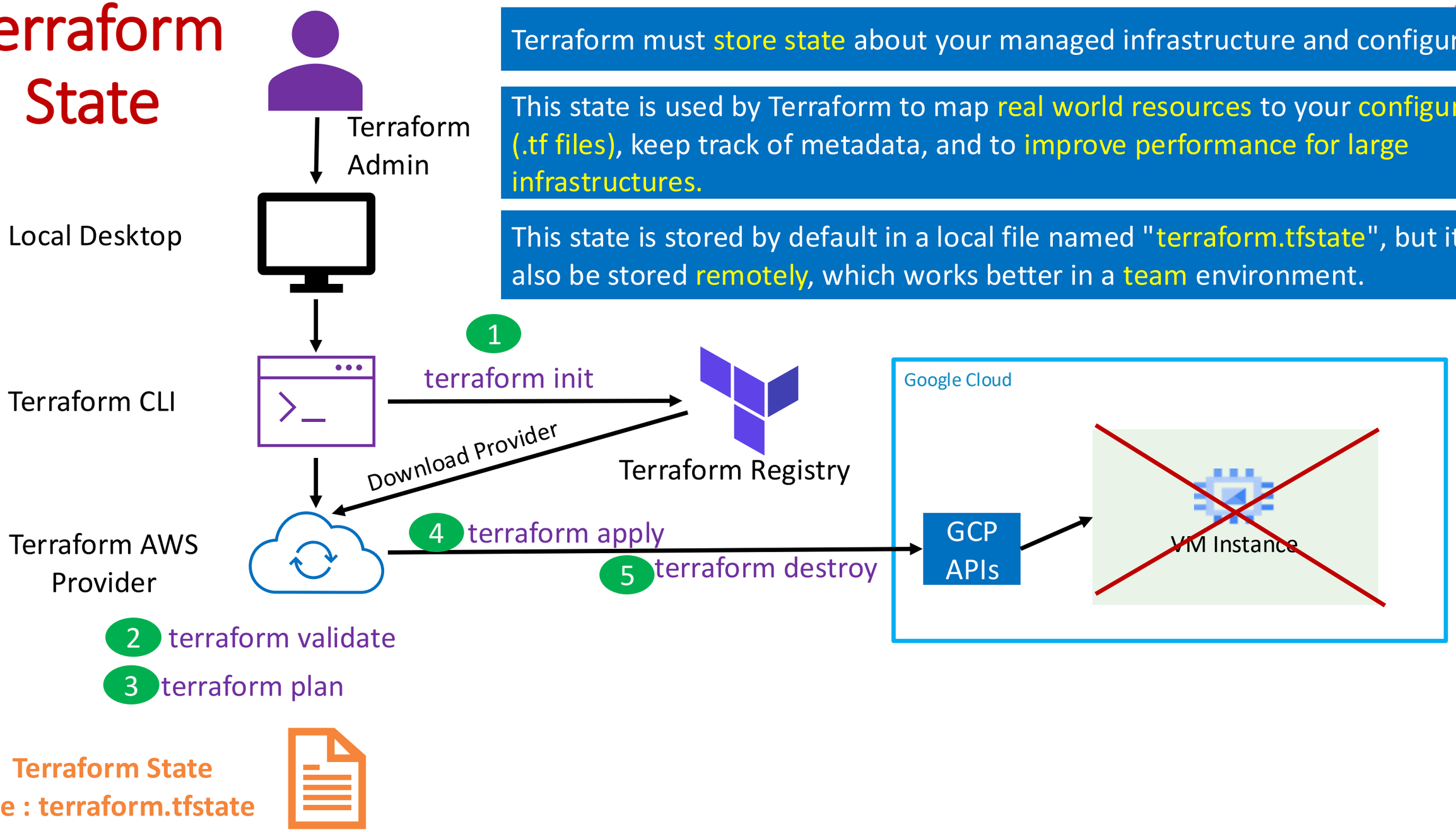
Terraform State

Terraform State

Terraform must store state about your managed infrastructure and configuration

This state is used by Terraform to map real world resources to your configuration (.tf files), keep track of metadata, and to improve performance for large infrastructures.

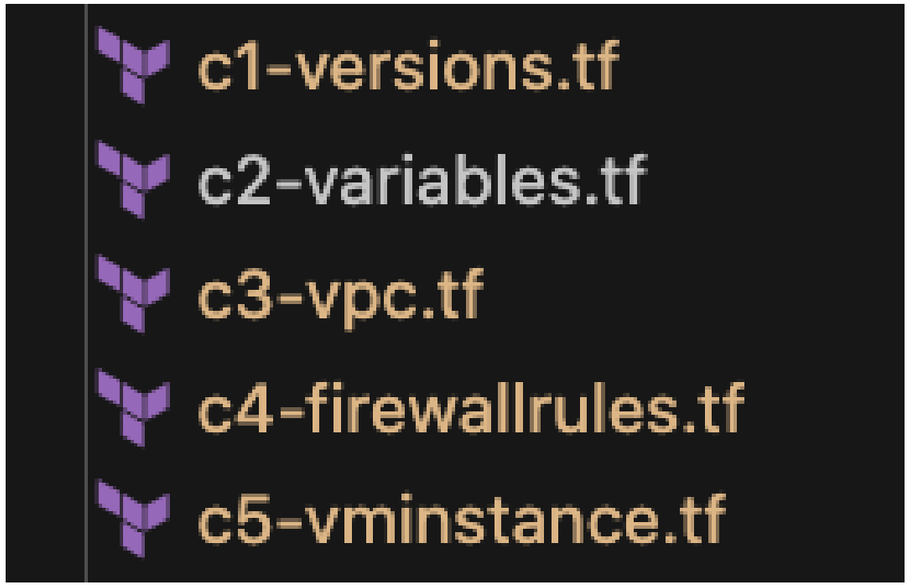
This state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.



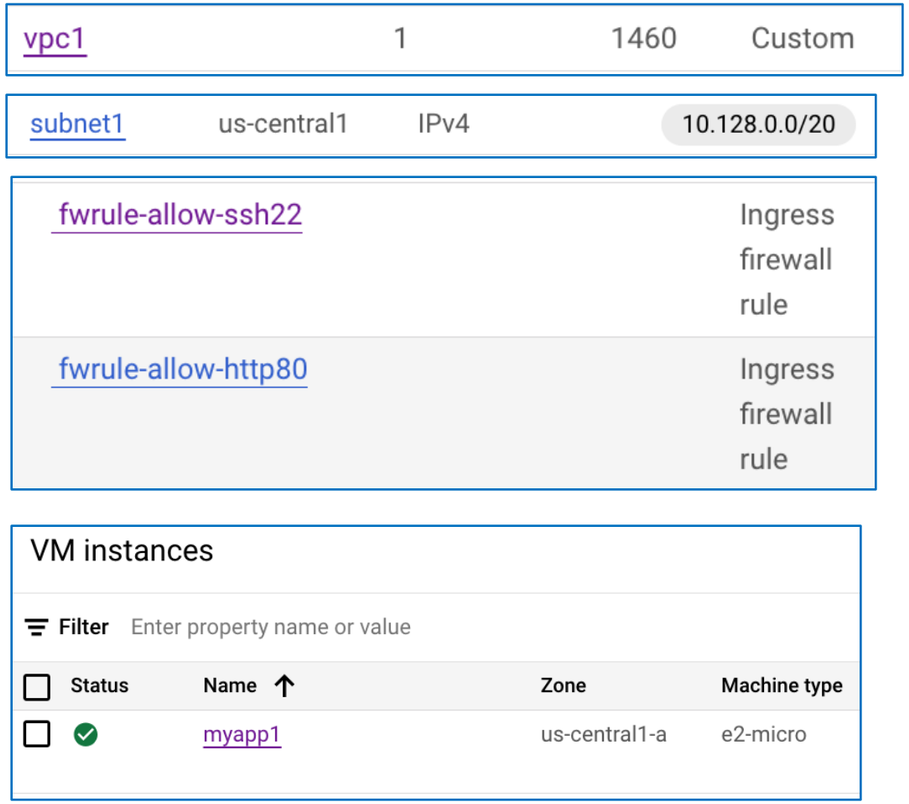
Desired & Current Terraform States

STACKSIMPLIFY

Terraform Configuration Files



Real World Resource
VM Instance, VPC, Subnet,
Firewall Rules



Desired State

Current State

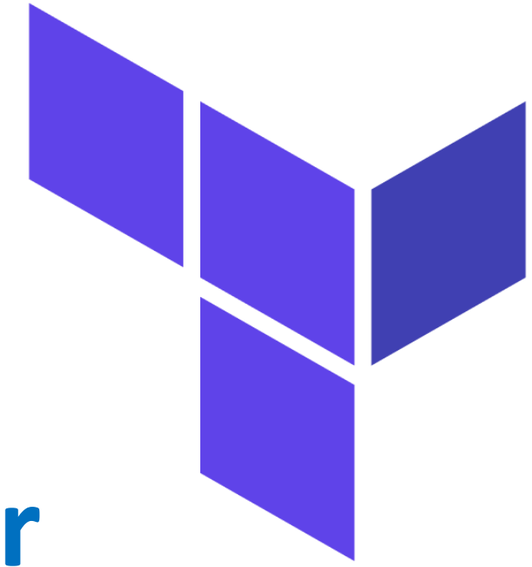
STACKSIMPLIFY

Demo-4

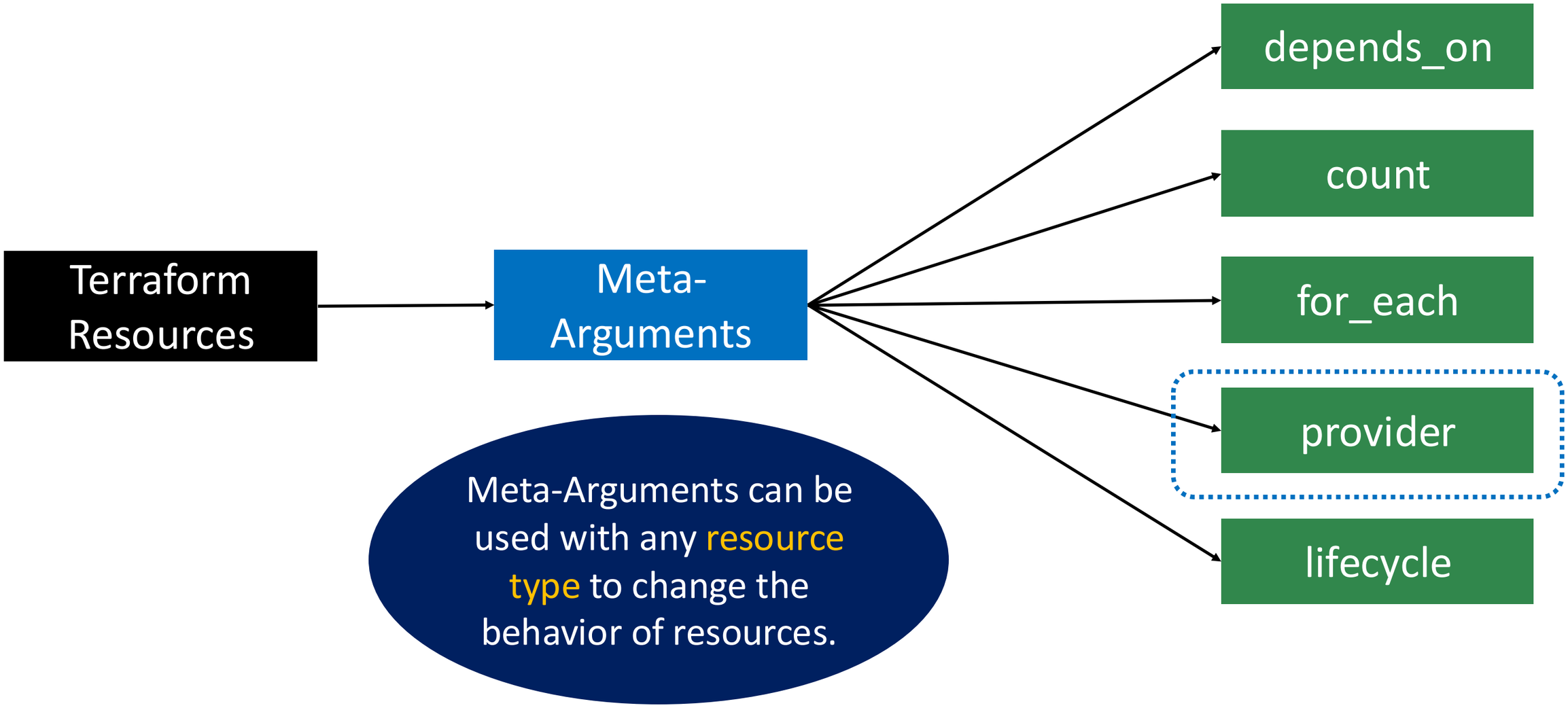
Terraform

Multiple Providers

Meta-Argument: provider



Resource Meta-Arguments



Terraform Multiple Providers

STACKSIMPLIFY

- We can define multiple configurations for the same provider (Ex: google) and select which one to use on a per-resource or per-module basis.
- The primary reason for this is to support multiple regions for a cloud platform, additionally for multiple environments (dev, prod) considering each environment has different provider configurations (Example: different authentication credentials for dev and prod)
- We can use the alternate provider in a resource, data or module by referencing it as <PROVIDER NAME>.<ALIAS>

```
# Terraform Provider-1: us-central1
provider "google" {
  project = "gcplearn9"
  region = "us-central1"
  alias = "us-central1"
}

# Terraform Provider-2: europe-west1
provider "google" {
  project = "gcplearn9"
  region = "europe-west1"
  alias = "europe-west1"
}
```

```
# Resource: Subnet1
resource "google_compute_subnetwork" "mysubnet1" {
  provider = google.us-central1  # Define provider to use
  name = "subnet1"
  ip_cidr_range = "10.128.0.0/20"
  network = google_compute_network.myvpc.id
}

# Resource: Subnet2
resource "google_compute_subnetwork" "mysubnet2" {
  provider = google.europe-west1  # Define provider to use
  name = "subnet2"
  ip_cidr_range = "10.132.0.0/20"
  network = google_compute_network.myvpc.id
}
```

Terraform Multiple Providers

- We can define multiple configurations for the same provider (Ex: google) and select which one to use on a per-resource or per-module basis.
- **Usage**
  - To support multiple regions for a cloud platform (we will practically implement this)
  - For supporting multiple environments (dev, prod) considering each environment has different provider configurations
    - **Example:** Different authentication credentials for dev and prod configured in their respective provider blocks
- We can use the alternate provider in a resource, data or module by referencing it as <PROVIDER NAME>.<ALIAS>

```
# Terraform Provider-1: us-central1
provider "google" {
  project = "gcplearn9"
  region = "us-central1"
  alias = "us-central1"
}

# Terraform Provider-2: europe-west1
provider "google" {
  project = "gcplearn9"
  region = "europe-west1"
  alias = "europe-west1"
}
```

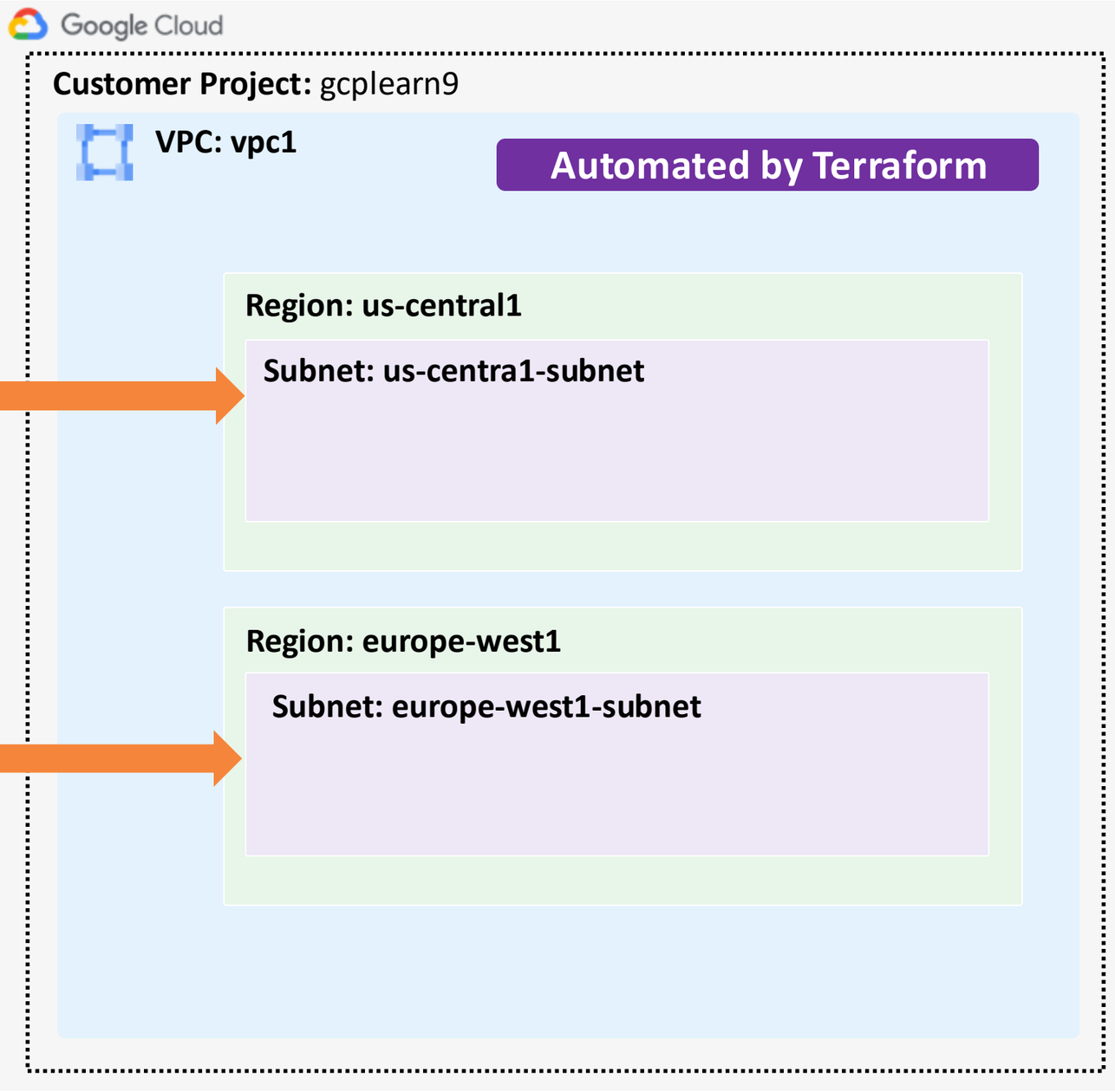
```
# Resource: Subnet1
resource "google_compute_subnetwork" "mysubnet1" {
  provider = google.us-central1   # Define provider to use
  name = "subnet1"
  ip_cidr_range = "10.128.0.0/20"
  network = google_compute_network.myvpc.id
}

# Resource: Subnet2
resource "google_compute_subnetwork" "mysubnet2" {
  provider = google.europe-west1   # Define provider to use
  name = "subnet2"
  ip_cidr_range = "10.132.0.0/20"
  network = google_compute_network.myvpc.id
}
```

Terraform Multiple Providers

STACKSIMPLIFY

```
# Terraform Provider-1: us-central1
provider "google" {
  project = "gcplearn9"
  region = "us-central1"
  alias = "us-central1"
}
```

```
# Terraform Provider-2: europe-west1
provider "google" {
  project = "gcplearn9"
  region = "europe-west1"
  alias = "europe-west1"
}
```



What are we going to learn?

Demo: Terraform Concept: Multiple Providers

C1 — Update multiple providers

C2 — Use provider alias in subnet Terraform resources

STACKSIMPLIFY

Demo-5

Terraform

Input Variables

Output Values

Demo: Terraform Concept: Input Variables and Output Values

Terraform Input Variables

- Input variables allow us to customize the Terraform Resources or modules without altering the source code.
- **Key Benefit:** Allows us to make the code reusable by just changing variables
- We have multiple options to declare variables
  - varibales.tf (Default value)
  - terraform.tfvars
  - vm.auto.tfvars
  - vm.tfvars
  - Environment Variables
  - --var-file flag
  - --var flag

variables.tf

```
# GCP Compute Engine Machine Type
variable "machine_type" {
  description = "Compute Engine Machine Type"
  type = string
  default = "e2-small"
}
```

terraform.tfvars

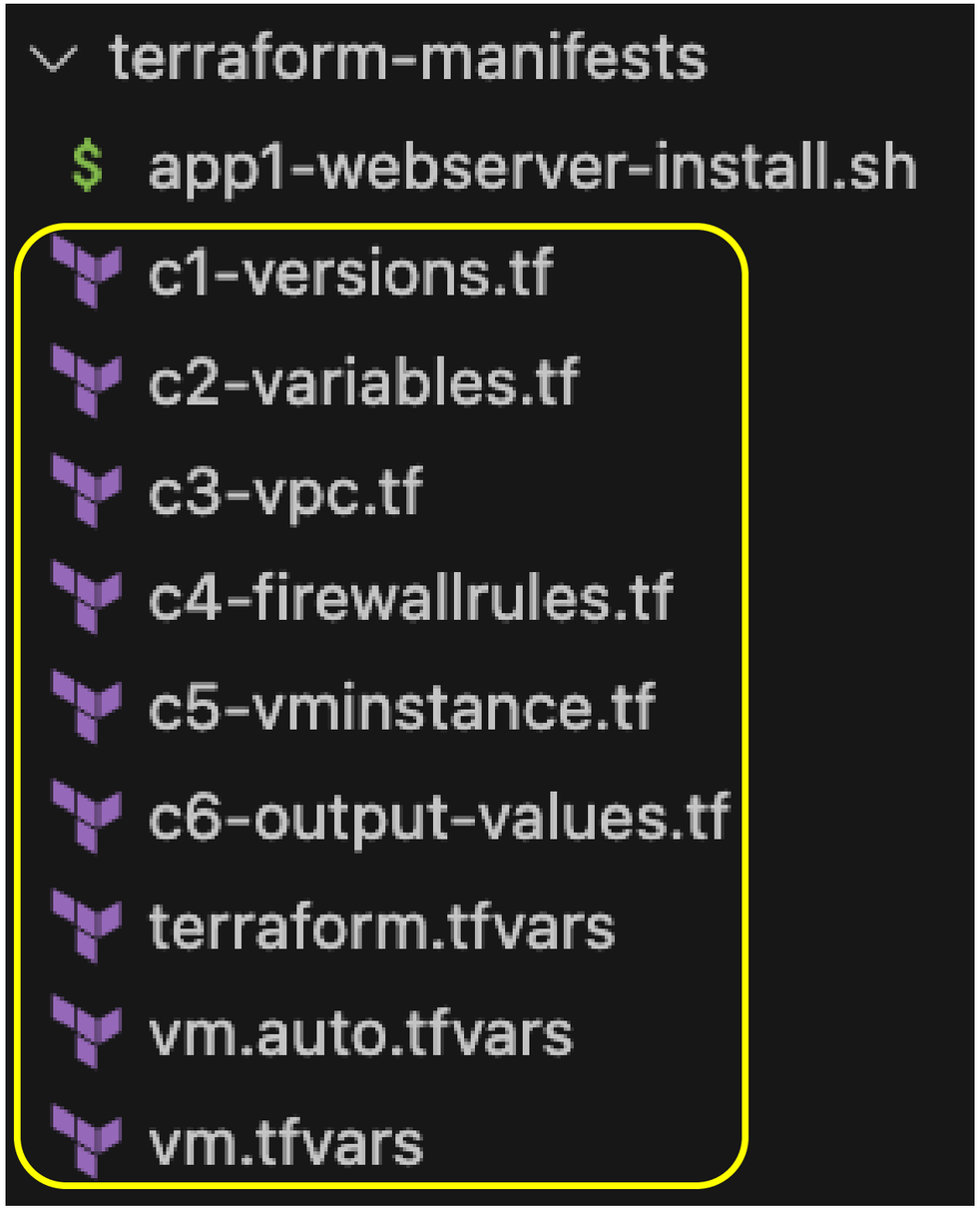
terraform.tfvars

terraform-on-google-cloud > 05-Terraform-Variables-Output-Values >

```
1  gcp_project   = "gcplearn9"
2  gcp_region1   = "us-central1"
3  machine_type  = "e2-micro"
```

What are we going to learn?

STACKSIMPLIFY



Demo: Terraform Concept: Input Variables

C2	Understand and define Terraform Input Variables
C1	Update Provider block with Terraform input variables
C3	Update subnet region with input variable
C5	Update VM Instance machine type using input variable
TF VARS	Play with terraform.tfvars, vm.auto.tfvars and vm.tfvars

Terraform Output Values

- Output values are like return values in programming languages.
- Output values provide about your infrastructure available on the command line
- With output values we can expose information for other Terraform configurations to use

Output Values

```
Outputs:

vm_external_ip = "34.45.238.213"
vm_id = "projects/gcplearn9/zones/us-central1-a/instances/myapp1"
vm_instanceid = "3515727697506246169"
vm_machine_type = "e2-micro"
vm_name = "myapp1"
vm_selflink = "https://www.googleapis.com/compute/v1/projects/gcplearn9/zones/us-central1-a/instances/myapp1"
```

Output Values - Attributes

STACKSIMPLIFY

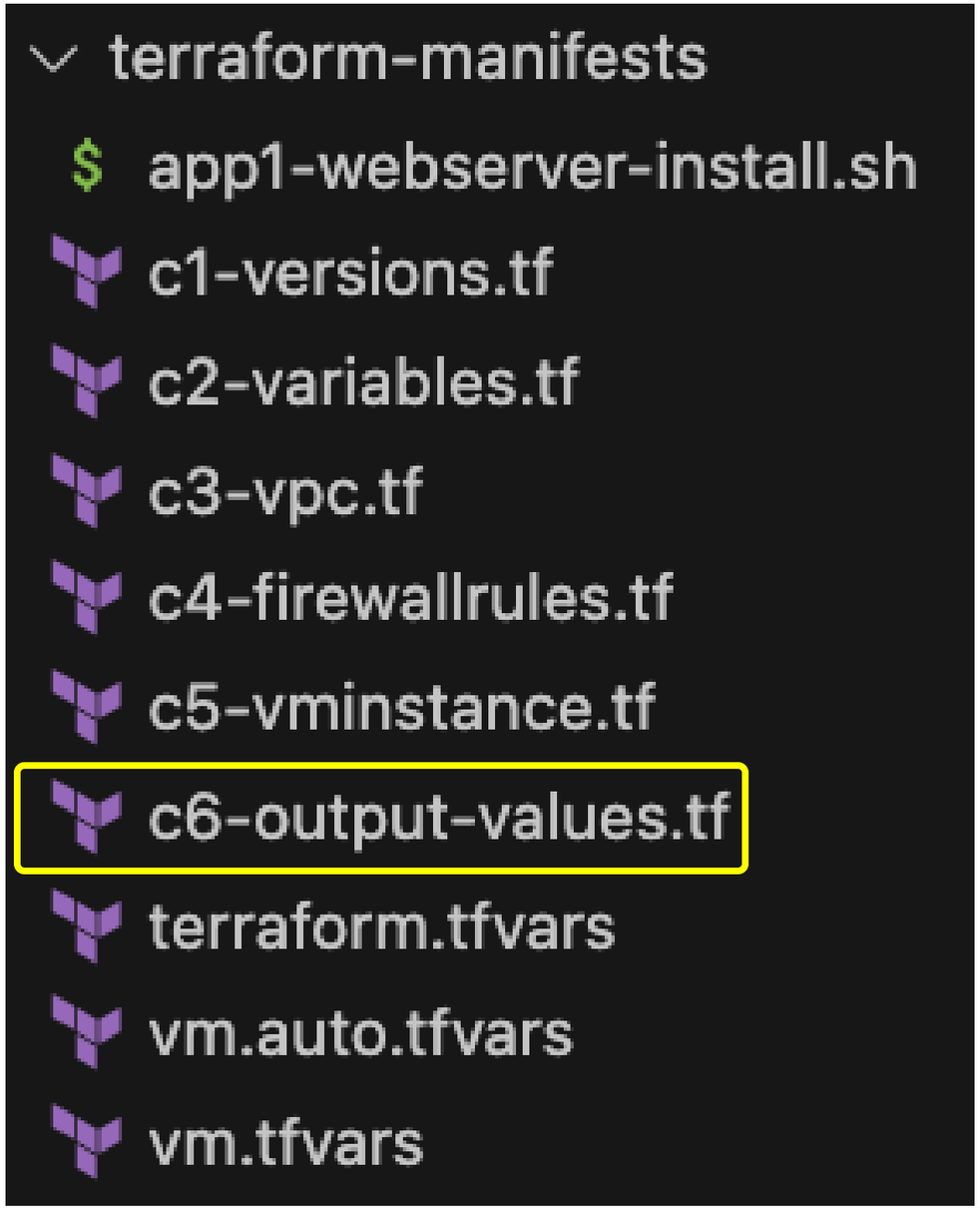
```
# Terraform Output Values
## ATTRIBUTES
output "vm_instanceid" {
  description = "VM Instance ID"
  value = google_compute_instance.myapp1.instance_id
}

output "vm_selflink" {
  description = "VM Instance Self link"
  value = google_compute_instance.myapp1.self_link
}
```

Output Values - Arguments

```
## ARGUMENTS
output "vm_name" {
  description = "VM Name"
  value = google_compute_instance.myapp1.name
}

output "vm_machine_type" {
  description = "VM Machine Type"
  value = google_compute_instance.myapp1.machine_type
}
```

What are we going to learn?

STACKSIMPLIFY



Demo: Terraform Concept: Output Values

C6

Understand and define Terraform Output values

STACKSIMPLIFY

Demo-6

Terraform Meta-Argument count

Demo: Terraform Concept: Meta-argument count

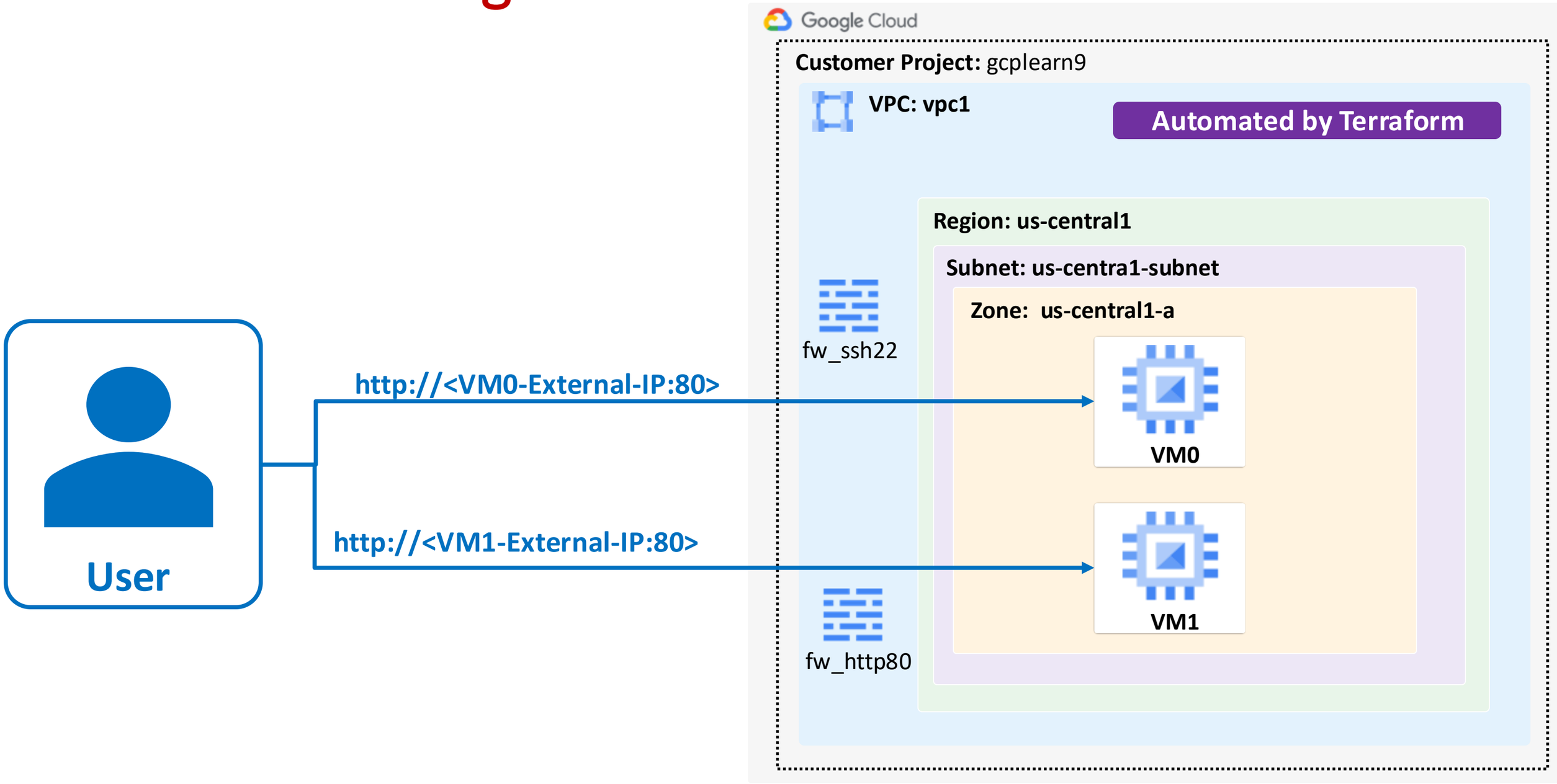
Terraform Meta-Argument: count

- By default, resource block creates a single instance of a resource
  - **For example:** 1 VM Instance
- With the meta-argument count we can create specified number of similar instances of a resource
  - **For example:** count = 2 (will create 2 VM instances from the same defined resource block)
- **count.index:** The distinct index number (starting with 0) corresponding to this instance.

Resource Meta-Argument: count

```
# Resource Block: Create a Compute Engine instance
resource "google_compute_instance" "myapp1" {
  # Meta-Argument: count
  count = 2
  name         = "myapp1-vm-${count.index}"
  machine_type = var.machine_type
  zone         = "us-central1-a"
```

Terraform Meta-argument: count



Terraform: For expression

- **For Expression:** Enables efficient management of multiple resources by looping through lists, maps, or sets
- **Examples:**
  - For loop with list
  - For loop with map
  - For loop with map advanced
  - Legacy splat operator
  - Latest generalized splat operator

For expression

```
# Output - For Loop with List
output "for_output_list" {
  description = "For Loop with List"
  value = [for instance in google_compute_instance.myapp1: instance.name]
}

# Output - For Loop with Map
output "for_output_map1" {
  description = "For Loop with Map"
  value = {for instance in google_compute_instance.myapp1: instance.name => instance.instance_id}
}

# Output - For Loop with Map Advanced
output "for_output_map2" {
  description = "For Loop with Map - Advanced"
  value = {for c, instance in google_compute_instance.myapp1: c => instance.name}
}
```

```
# Output Legacy Splat Operator (Legacy) - Returns the List
output "legacy_splat_instance" {
  description = "Legacy Splat Operator"
  value = google_compute_instance.myapp1.*.name
}

# Output Latest Generalized Splat Operator - Returns the List
output "latest_splat_instance" {
  description = "Generalized latest Splat Operator"
  value = google_compute_instance.myapp1[*].name
}
```

What are we going to learn?

Demo: Terraform Concept: Meta-argument count

NC	No Changes from C1 to C4
C5	Define VM Instance with Meta-argument count
C6	Define Terraform output values with Terraform for expression and also learn about Splat operator

Demo-7

Terraform

Data sources

Demo: Terraform Concept: Datasources

Terraform Data sources

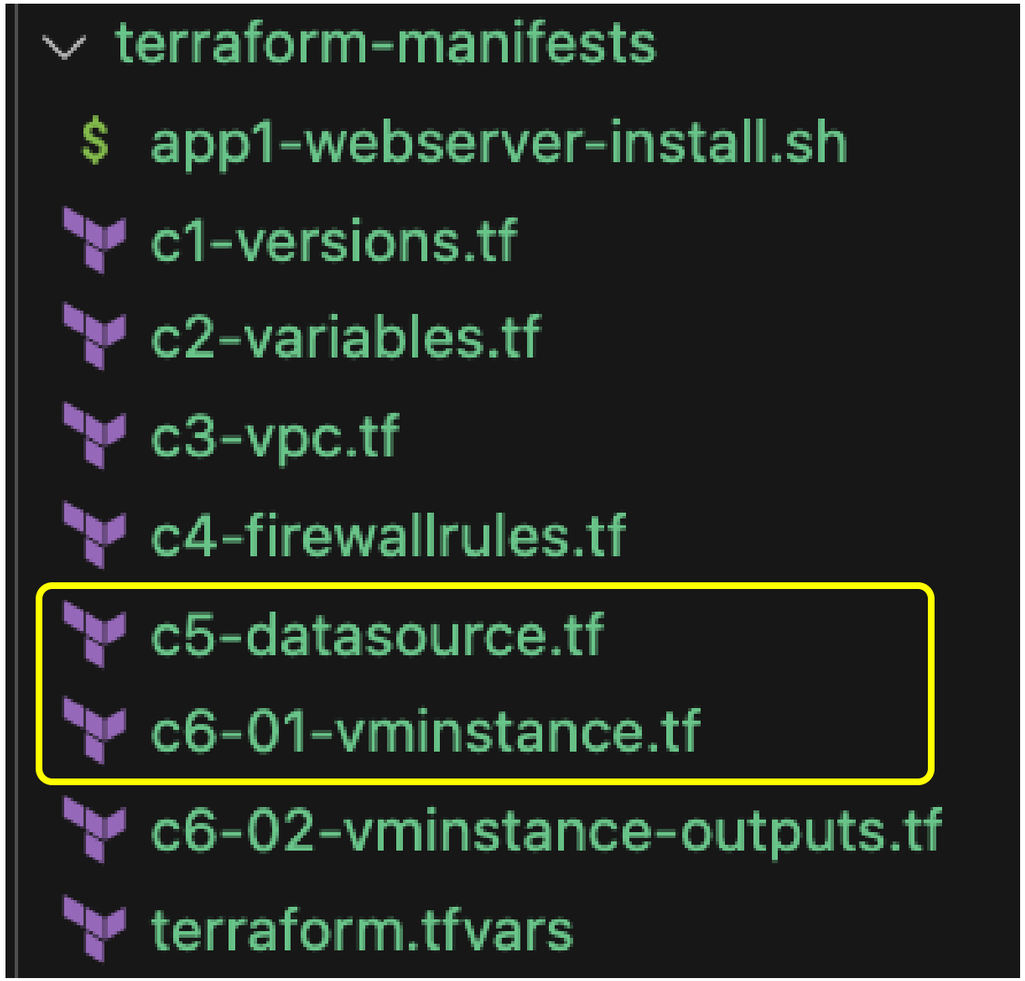
- **Data sources:** allow Terraform to use information defined outside of Terraform
- A data source is accessed via a special kind of resource known as a data resource, declared using a data block:
- **Use case:**
  - **Step-01:** Get the available compute zones in that respective region using google_compute_zone datasource
  - **Step-02:** Create two VM Instances in two available compute zones using the Meta-argument count

Terraform Data source

```
# Terraform Datasources
/* Datasource: Get a list of Google
Compute zones that are UP in a region */
data "google_compute_zones" "available" {
  status = "UP"
}

# Output value
output "compute_zones" {
  description = "List of compute zones"
  value = data.google_compute_zones.available.names
}
```

What are we going to learn?

STACKSIMPLIFY



Demo: Terraform Concept: Datasources

NC	No Changes from C1 to C4
C5	Define a data source to get a list of all available zones in that region
C6-01	Update VM Instance with zone value updated from Terraform data source

STACKSIMPLIFY

Demo-8

Terraform Meta-Argument for_each

Demo: Terraform Concept: Meta-Argument for_each

Terraform Meta-Argument: for_each

- By default, resource block creates a single instance of a resource
  - **For example:** 1 VM Instance
- The for_each meta-argument accepts a map or a set of strings and creates an instance for each item in that map or set
  - **For example:** Creates 1 VM instance for each compute zone
- **Each instance has a**
  - distinct infrastructure object associated with it
  - each is separately created, updated, or destroyed when the configuration is applied

Resource Meta-Argument: for_each

```
# Resource Block: Create a Compute Engine instance
resource "google_compute_instance" "myapp1" {
  # Meta-Argument: for_each
  for_each = toset(data.google_compute_zones.available.names)
  name         = "myapp1-vm-${each.key}"
  machine_type = var.machine_type
  zone         = each.key # You can also use each.value because for list
  tags         = [tolist(google_compute_firewall.fw_ssh.target_tags)[0],
```

- **each.key:** The map key (or set member) corresponding to this instance.
- **each.value:** The map value corresponding to this instance (If a set was provided, each.key == each.value)

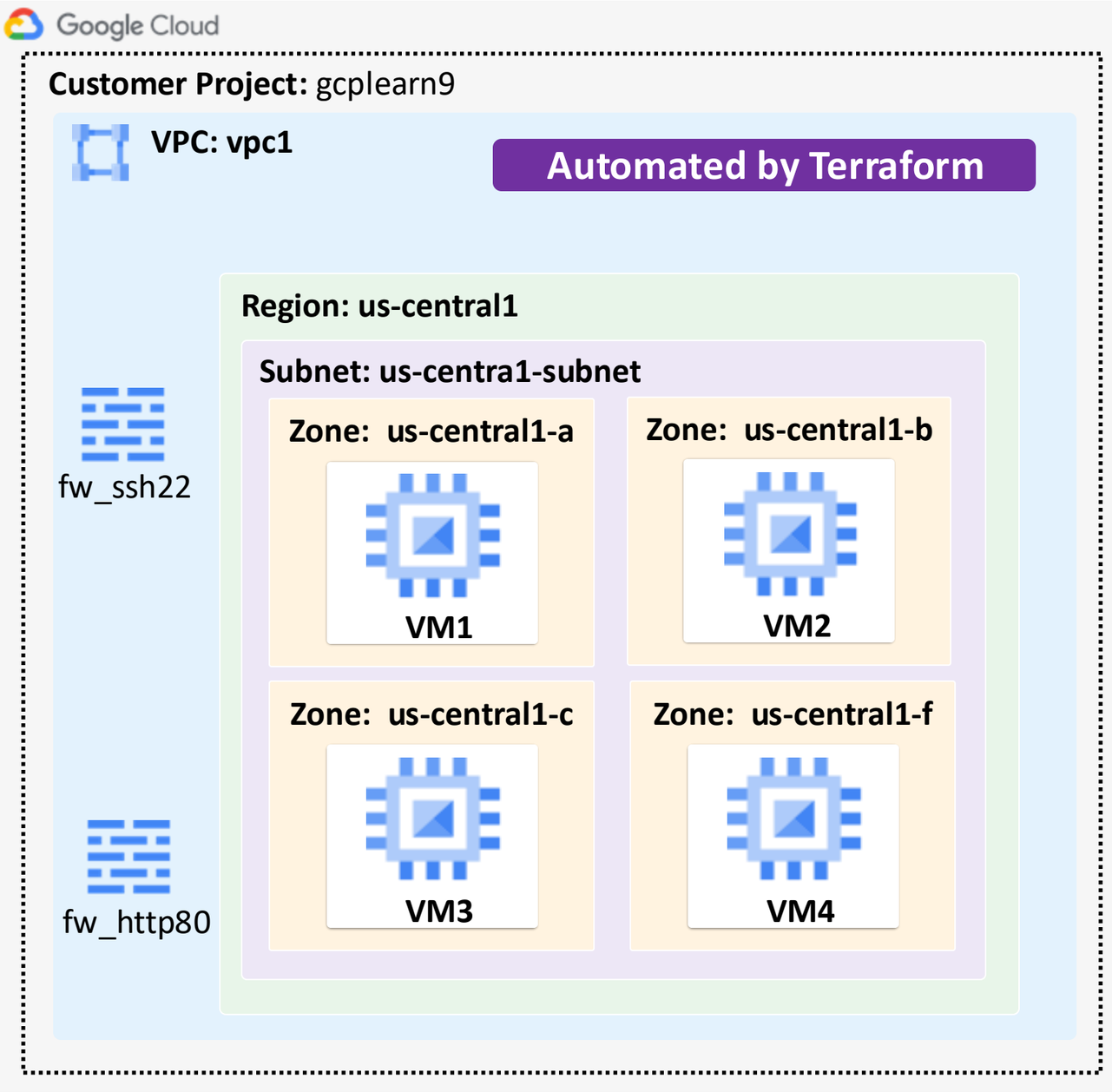
Terraform Meta-argument: for_each

STACKSIMPLIFY

Demo-1: for_each with Set values

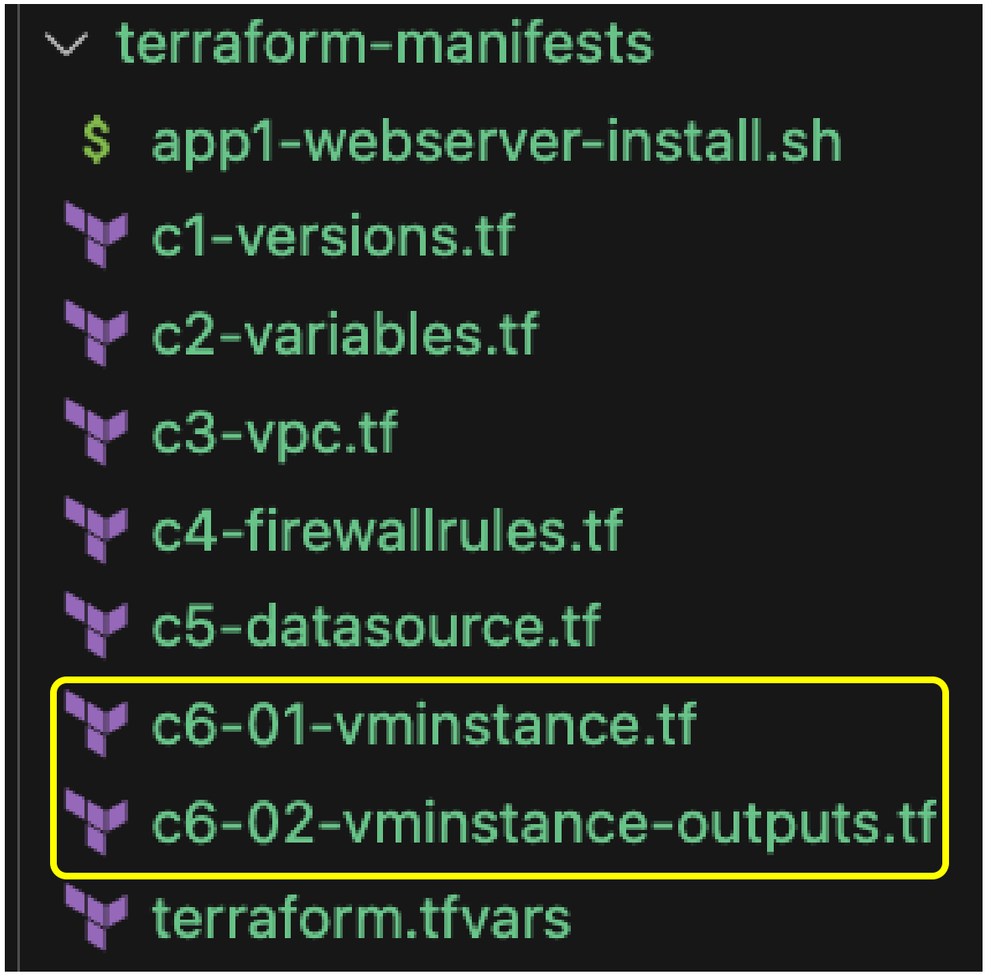
Creates VM instance for each available compute zone in a region

```
# Resource Block: Create a Compute Engine instance
resource "google_compute_instance" "myapp1" {
  # Meta-Argument: for_each
  for_each = toset(data.google_compute_zones.available.names)
  name         = "myapp1-vm-${each.key}"
  machine_type = var.machine_type
  zone         = each.key # You can also use each.value because
```



What are we going to learn?

STACKSIMPLIFY



Demo-01: Terraform Concept: Meta-argument for_each

NC	No Changes from C1 to C5
C6-01	VM Instance with for_each Meta-argument with Set Values
C6-02	Terraform outputs for VM Instance

Terraform Meta-argument: for_each

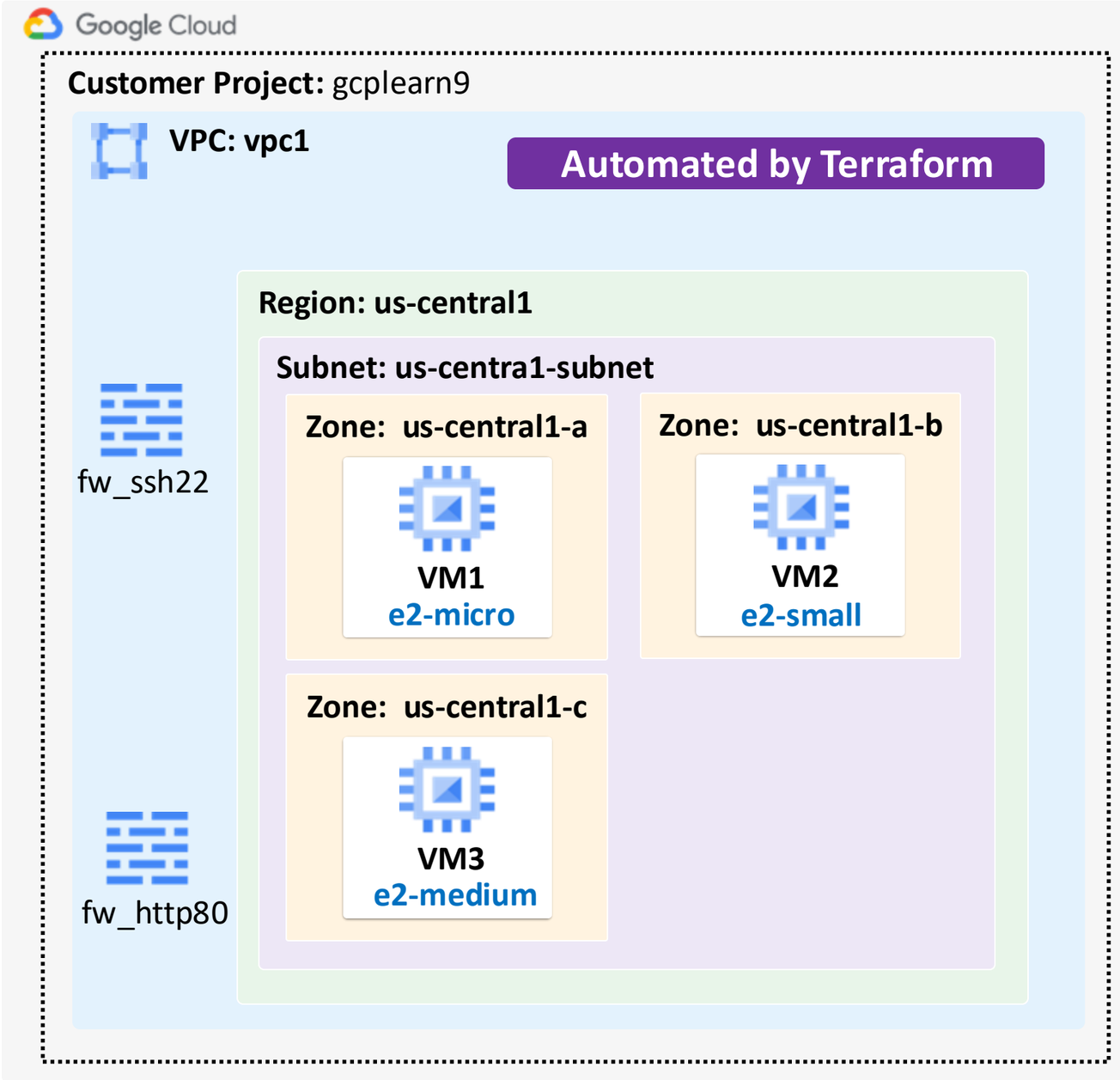
STACKSIMPLIFY

Demo-2: for_each with Map values

Creates VM instance with different machine type in each compute zone

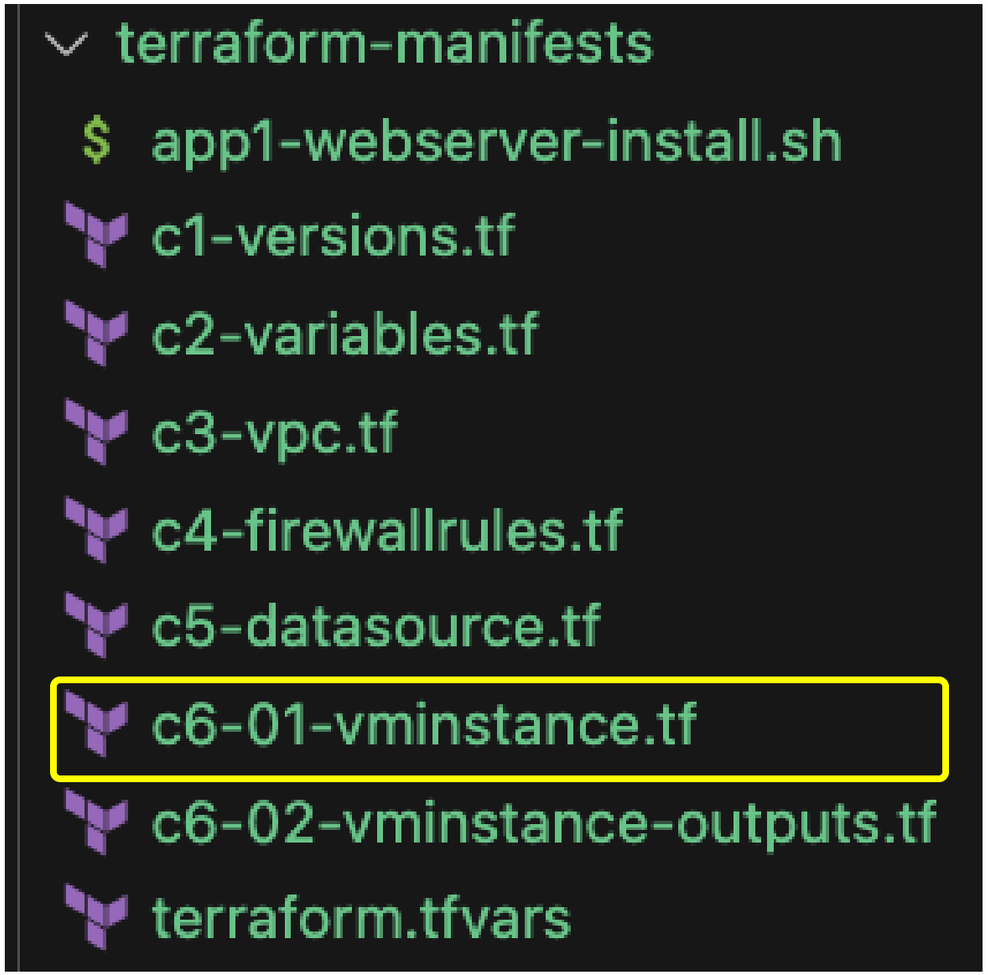
```
# Define a map with zone as key and machine_type as value
variable "zone_machine_map" {
  type = map(string)
  default = {
    "us-central1-a" = "e2-micro"
    "us-central1-b" = "e2-small"
    "us-central1-c" = "e2-medium"
  }
}
```

```
# Resource Block: Create a Compute Engine instance
resource "google_compute_instance" "myapp1" {
  # Meta-Argument: for_each
  for_each = var.zone_machine_map
  name         = "myapp1-vm-${each.key}"
  machine_type = each.value
  zone         = each.key
```



What are we going to learn?

STACKSIMPLIFY



Demo-02: Terraform Concept: Meta-argument for_each

NC	No Changes from C1 to C5
C6-01	VM Instance with for_each Meta-argument with Map Values

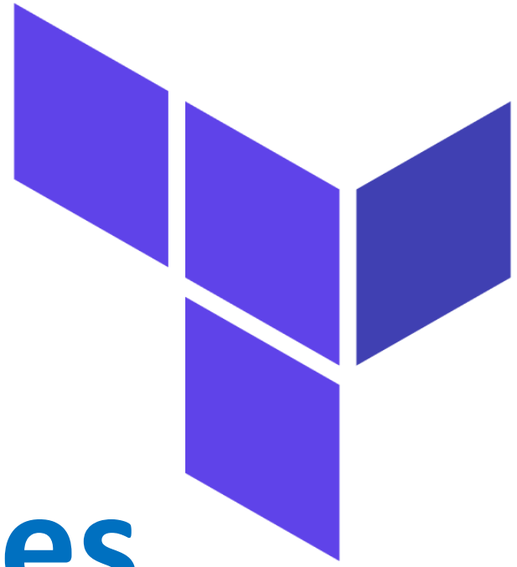
STACKSIMPLIFY

Demo-9

Terraform

Local Values

GCP Instance Templates



Demo: Custom VPC + Instance Template + VM Instance

Terraform Local Values

- **Local Values:** Assigns a name to an expression, so you can use the name multiple times within a module instead of repeating the expression
- Local values are like a function's temporary local variables
- **Usage:**
  - local.name
  - local.owners
- **Important Note:** Overusing local values can make the configuration hard to read for future maintainers by hiding the actual values used (use it wisely based on need)

Local Values

```
# Define Local Values in Terraform
locals {
  owners = var.business_divsion
  environment = var.environment
  name = "${var.business_divsion}-${var.environment}"
  #name = "${local.owners}-${local.environment}"
  common_tags = {
    owners = local.owners
    environment = local.environment
  }
}
```

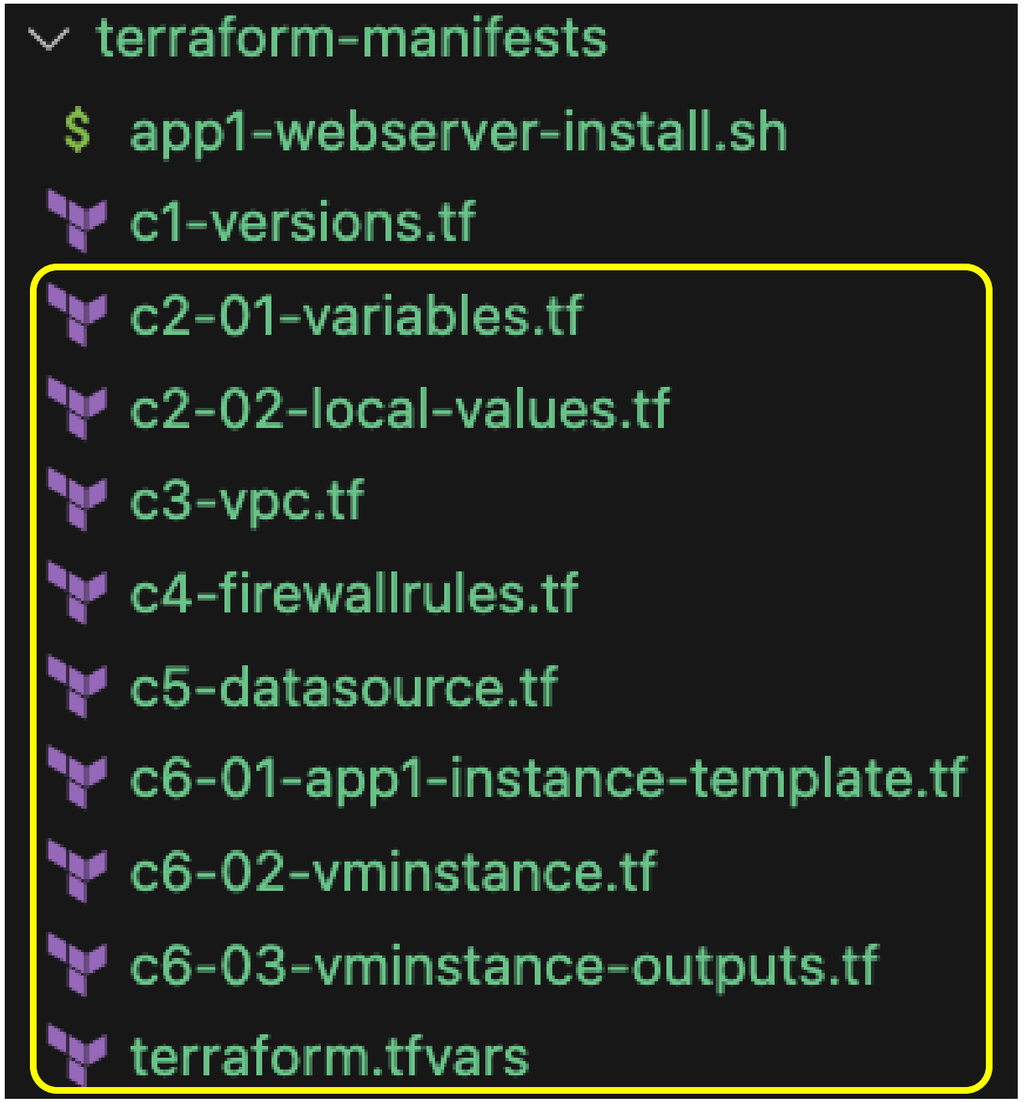
Expression

Using Local values in Resources

```
# Resource: VPC
resource "google_compute_network" "myvpc" {
  name = "${local.name}-vpc"
  auto_create_subnetworks = false
}

# Resource: Subnet
resource "google_compute_subnetwork" "mysubnet" {
  name = "${var.gcp_region1}-subnet"
  region = var.gcp_region1
  ip_cidr_range = "10.128.0.0/20"
  network = google_compute_network.myvpc.id
}
```

What are we going to learn?

STACKSIMPLIFY



Demo: Custom VPC + Instance Template + VM Instance

NC	No Changes in C1
C2-01	Update Variables required for local values (environment, business division) and also tfvars file
C2-02	Terraform Local Values concept + Update local values for all applicable terraform resources
C5	Add new data source google_compute_image to get image name dynamically (NO HARD CODING)
C6-01	Regional Instance Template
C6-02	VM Instance using Regional Instance Template
C6-03	Terraform outputs for VM Instance

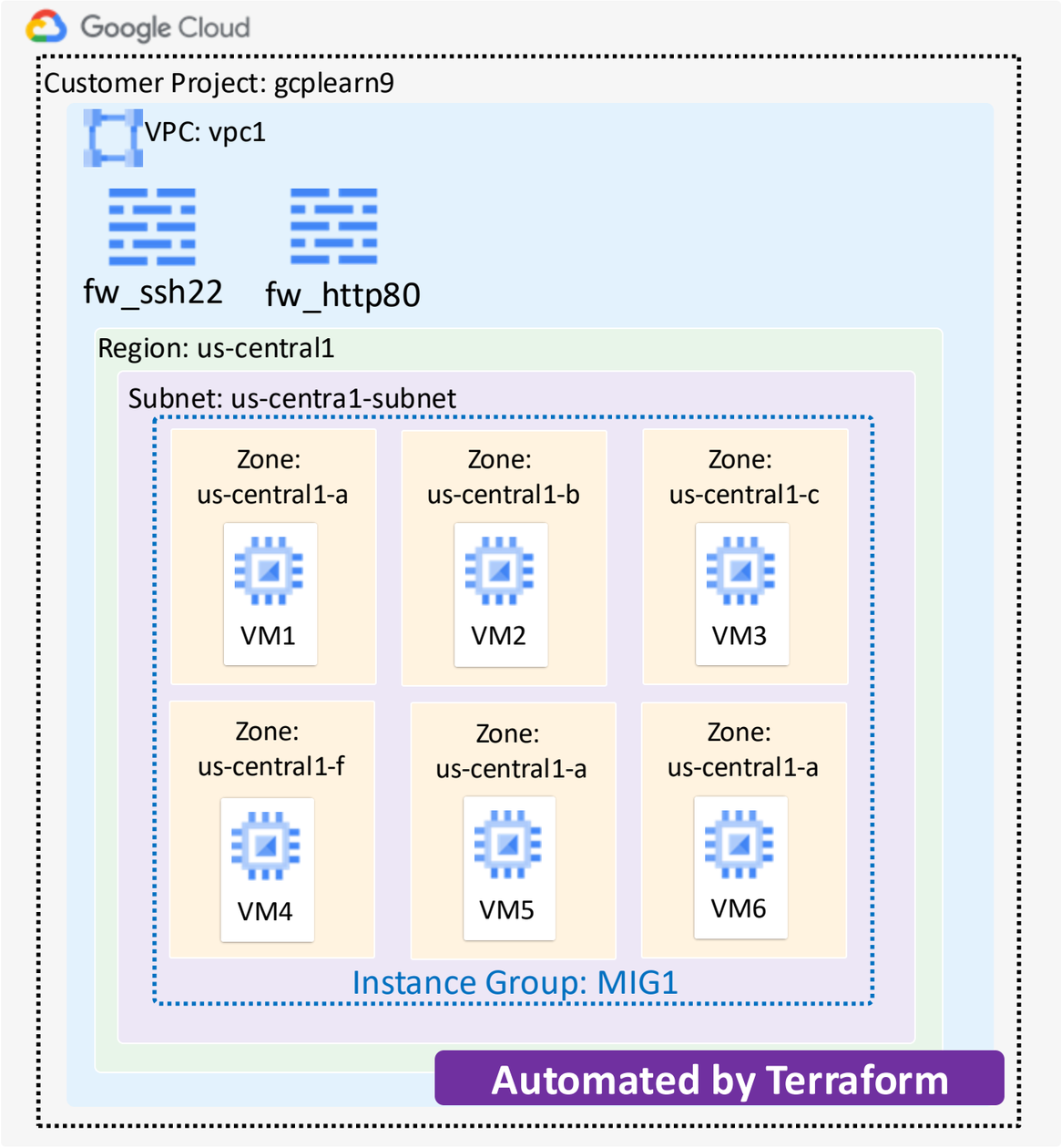
Demo-10

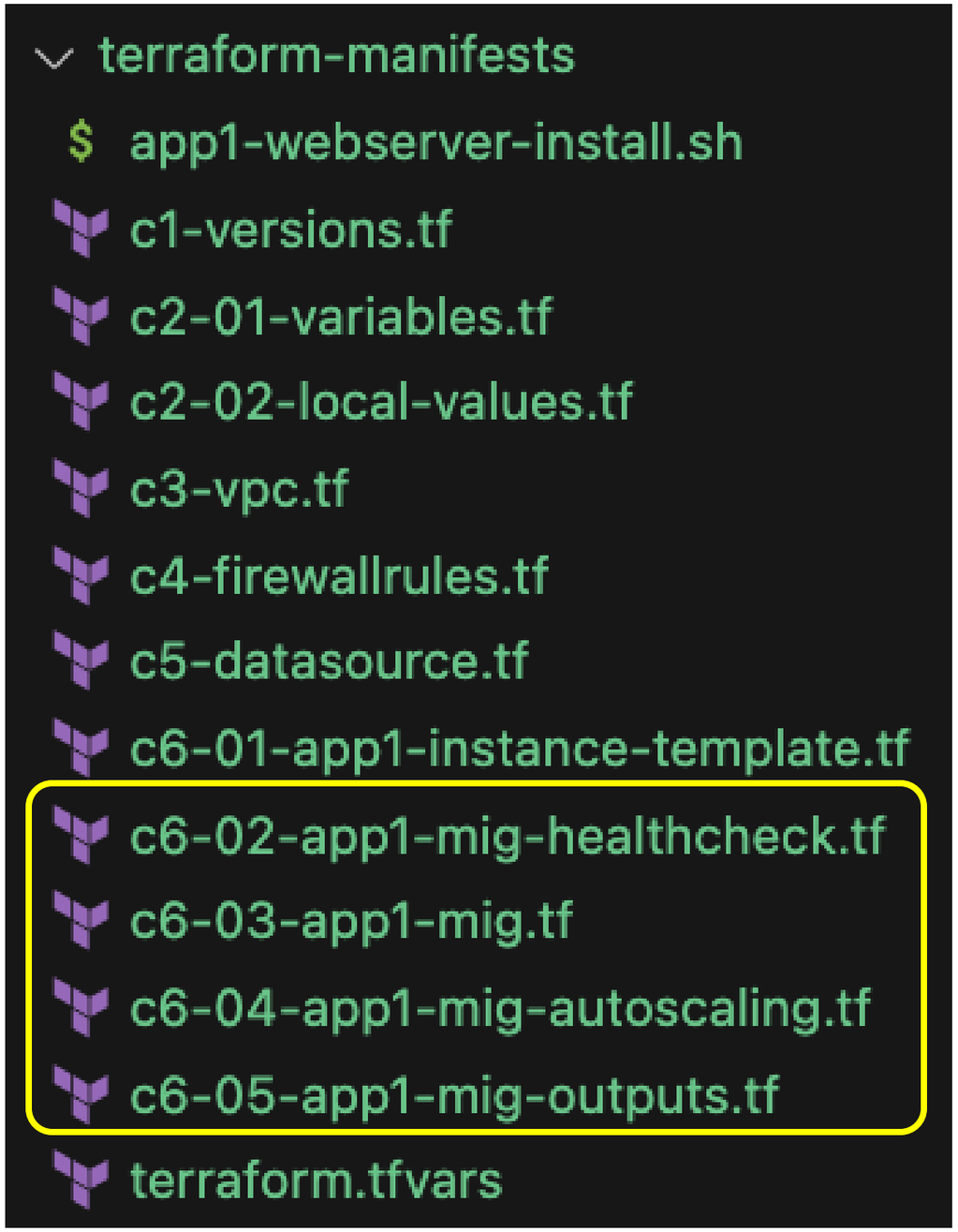
Terraform
GCP
Managed Instance Groups

Demo: Custom VPC + Regional MIG Public IP

Managed Instance Groups

- **What is an Instance Group ?**
  - Group of VM Instances managed as a single entity is called Instance Group
- **How many types of Instance Groups available in GCP ?**
  - Unmanaged Instance Groups
  - Managed Instance Groups(MIG) - Stateless
  - Managed Instance Groups(MIG) - Stateful
- **What is a Zonal MIG ?**
  - VM Instances created by MIG will be restricted to specific SINGLE ZONE
- **What is a Regional MIG?**
  - VM Instances created by MIG will be distributed across selected Zones
  - Regional MIG gives HIGH AVAILABILITY (RECOMMENDED)





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Public IP

NC	No Changes from C1 to C6-01
C6-02	Regional Health Check for Managed Instance Group
C6-03	Managed Instance Group Terraform Resource
C6-04	MIG Autoscaling Terraform Resource
C6-05	Terraform Outputs for MIG

STACKSIMPLIFY

Demo-11

Terraform

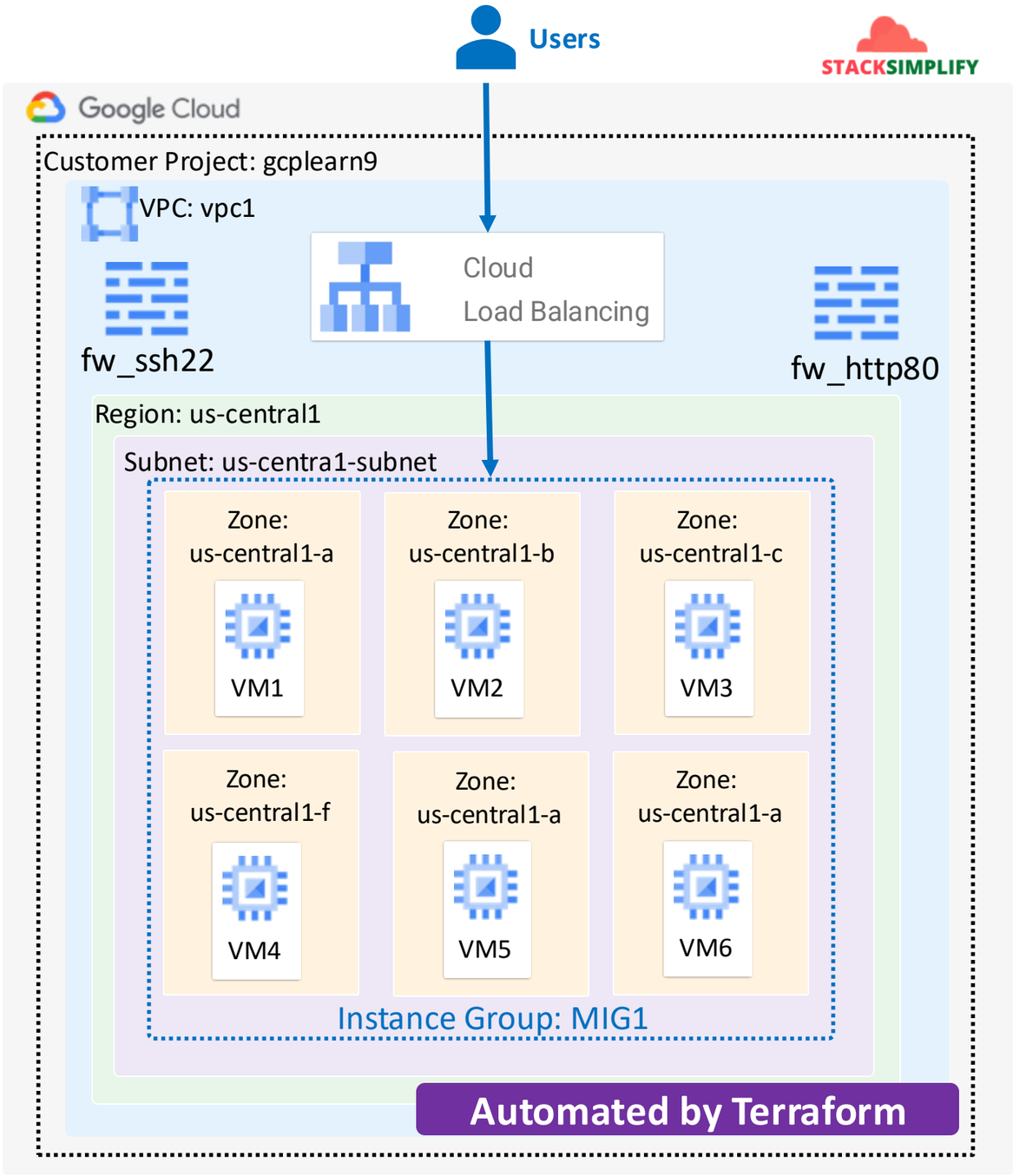
GCP

Regional Application Lod Balancer

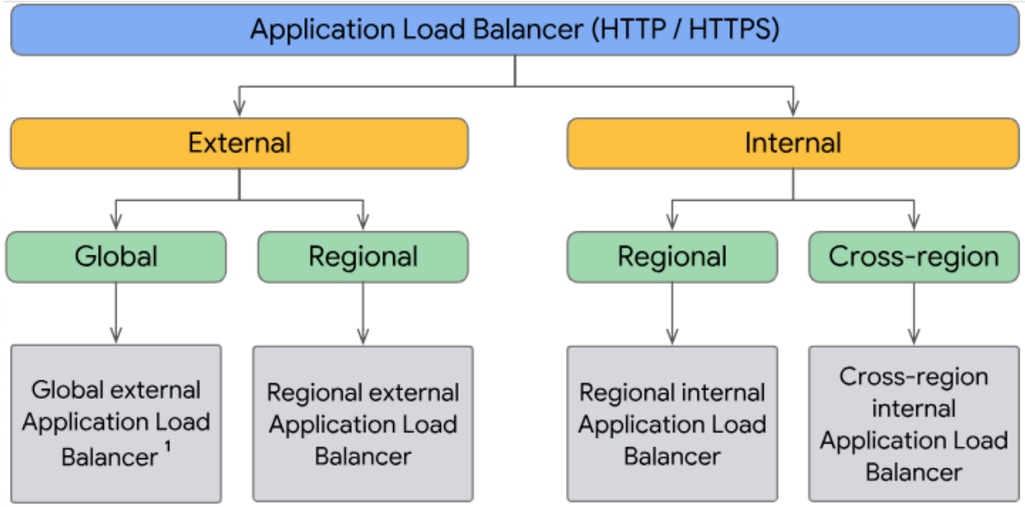
Demo: Custom VPC + Regional MIG Public IP +
Regional Application Load Balancer HTTP

GCP Cloud Load Balancing

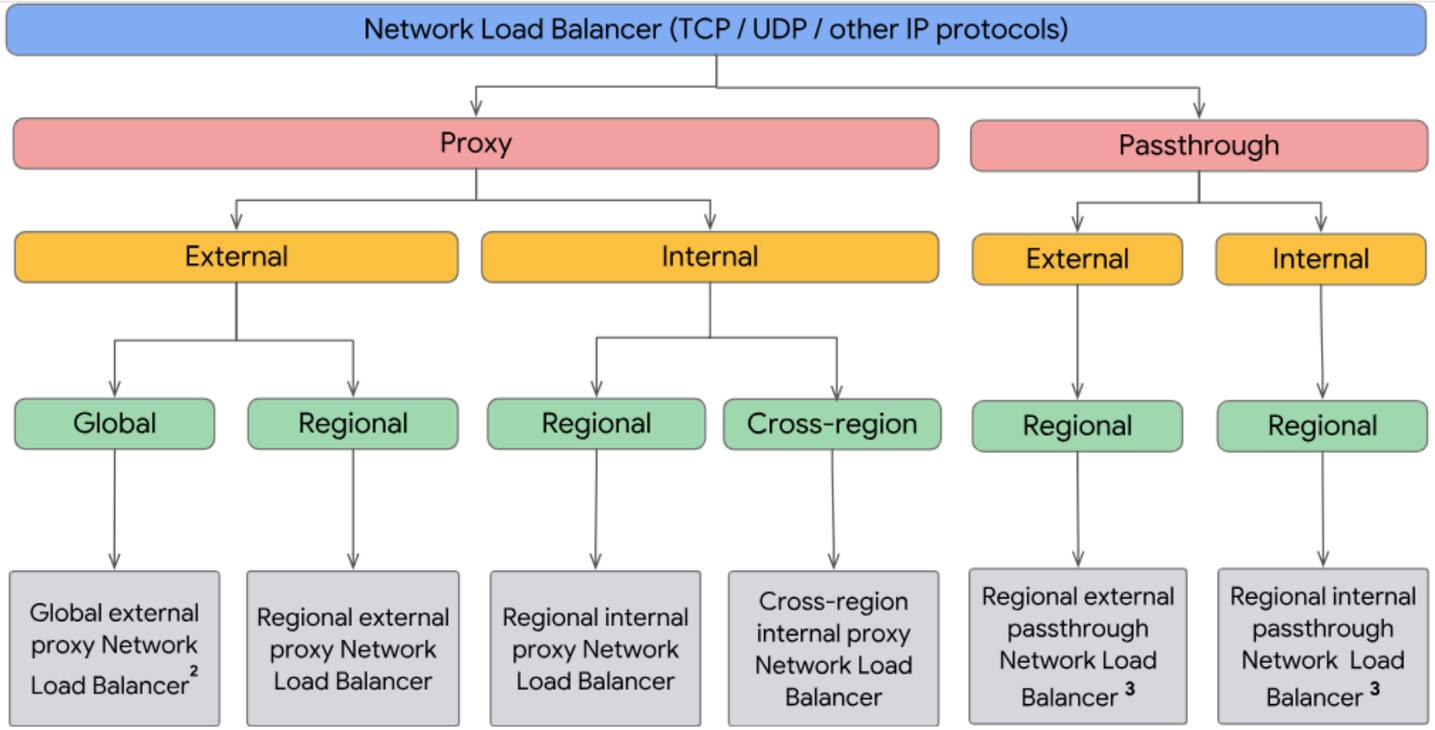
- **Cloud Load Balancing:** Distributes user traffic across multiple instances of your applications
- Primarily used for providing **HIGH AVAILABILITY** for your applications
- Fully **distributed** managed service
  - **Global:** load balance across regions
  - **Regional:** load balancer across zones in a region
- **Layer7 Load Balancer**
  - Application load balancer
    - HTTP or HTTPS
- **Layer4 Load Balancer**
  - Network Load Balancers - Proxy
    - TCP (TCP Proxy)
    - TCP with optional SSL offload (SSL Proxy)
  - Network Load Balancers - Pass-through
    - TCP, UDP, ICMP, ICMPv6, SCTP, ESP, AH, and GRE



Google Cloud Load Balancing - Types



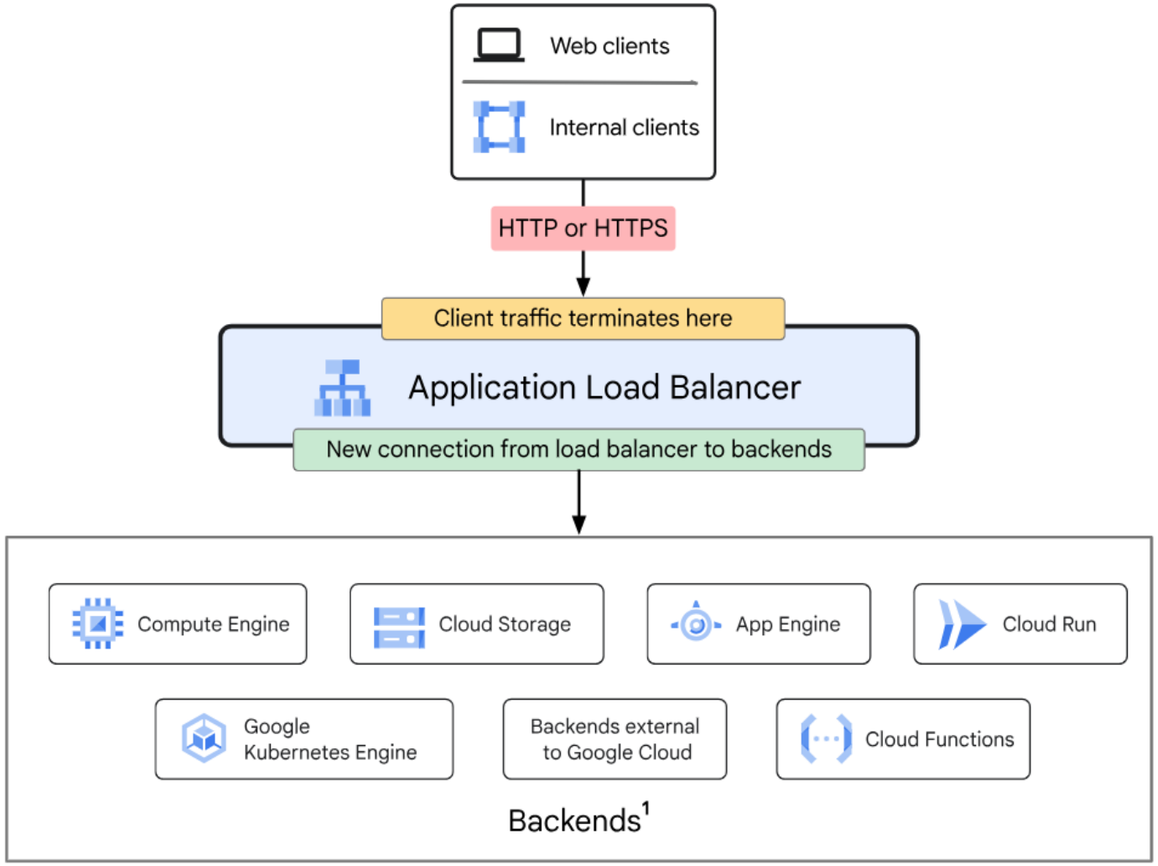
Reference: https://cloud.google.com/static/load-balancing/images/lb-product-tree.svg



- We can create approximately 10 types of Load Balancers
- On a very high-level, these are categorized as
  - Application Load Balancers
  - Network Load Balancers - Proxy
  - Network Load Balancers - Pass-through

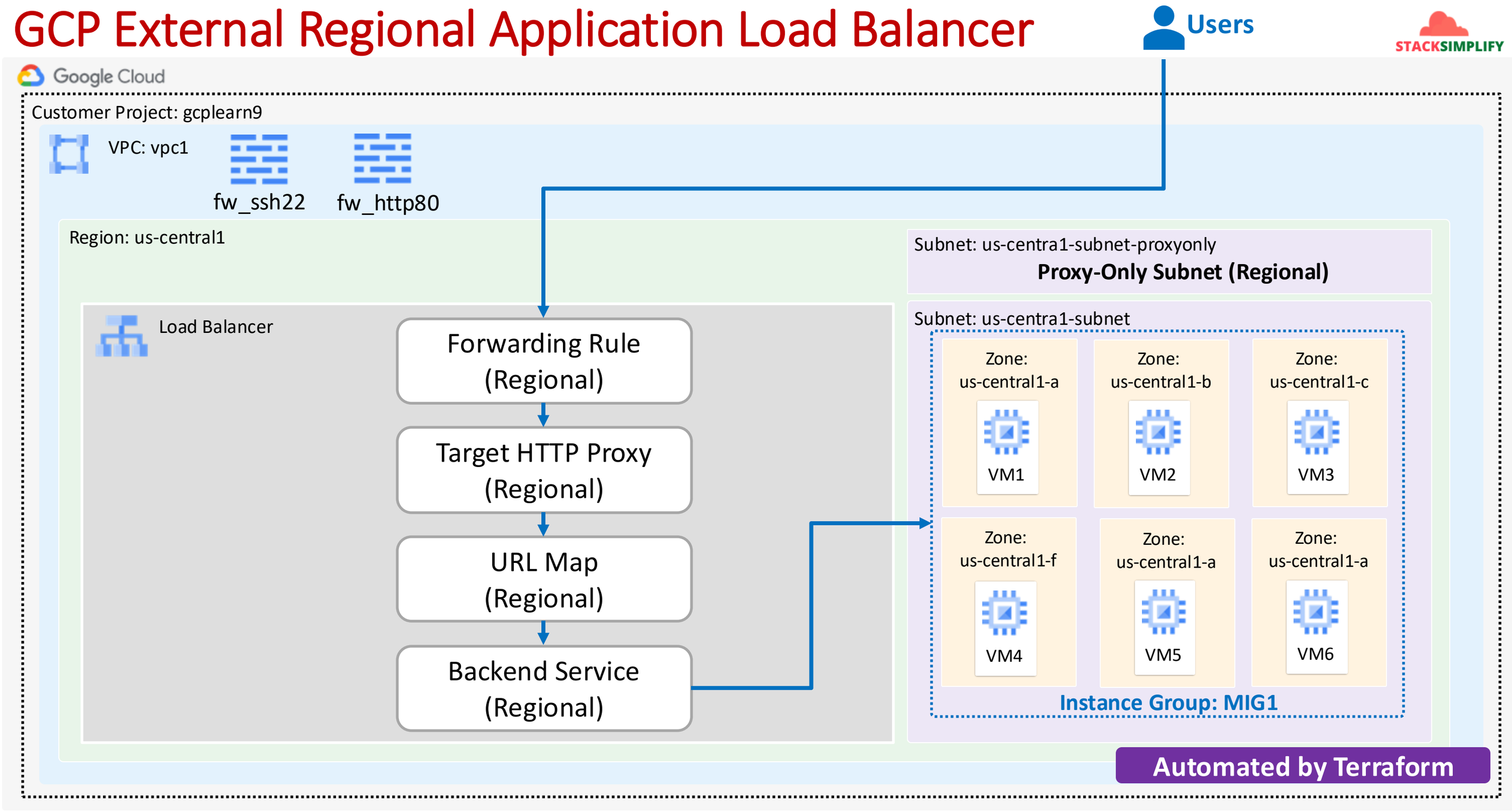
Google Cloud Load Balancing - Terminology

- **Backend**
  - Google Cloud services that receive traffic from Load Balancer
    - Instance Groups
    - Cloud Storage
    - App Engine
    - Cloud Run
    - GKE
    - Cloud Functions
    - Backends external to Google Cloud
- **Frontend**
  - Define Load Balancer IP address, port, protocol (HTTP, HTTPS, TCP, UDP, SSL etc)
  - We will use this IP address to access the load balancer



Reference: https://cloud.google.com/static/load-balancing/images/application-load-balancer.svg

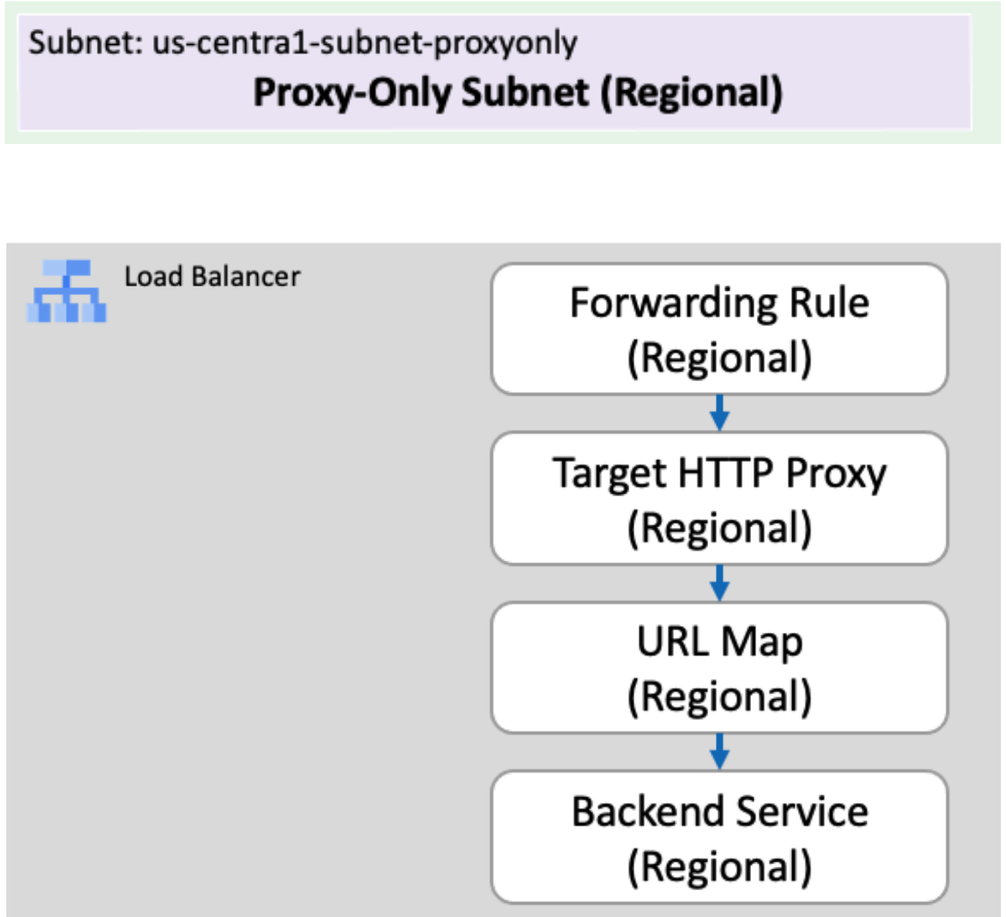
GCP External Regional Application Load Balancer



Google Cloud Load Balancing - Terminology

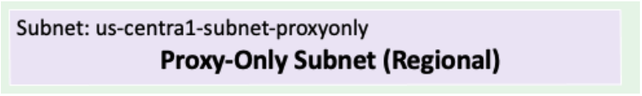
STACKSIMPLIFY

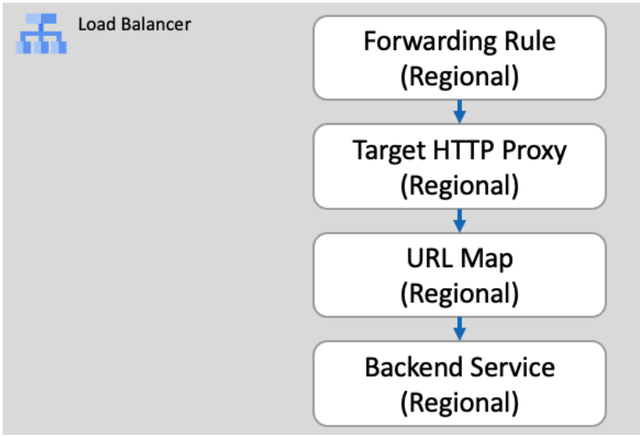
- **Proxy-Only Subnet**
  - Uses Envoy proxies which manage traffic routing and security
  - Provides dedicated IP Address pool
  - **Routing:** These proxies terminate connections from clients and handle traffic routing based on factors like URL Maps
    - HTTP Path routing
    - Host header routing
    - Custom Header routing and many more
  - **SSL:** Proxy layer can perform advanced security functions like
    - SSL Termination
  - **Scalability:** Can be automatically scaled by GCP to meet traffic demands



Google Cloud Load Balancing - Terminology

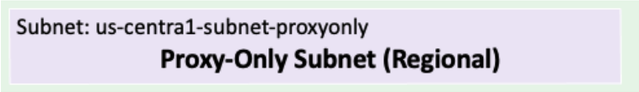
- **Forwarding Rule:**
  - Routes incoming traffic based on pre-defined criteria like IP address, port, and protocol.
  - It directs traffic to a specific load balancing configuration.
  - **Benefits:**
    - Provides a single, easy-to-manage IP address for your application in DNS records.
- **Target HTTP(S) Proxy:**
  - Terminates client connections and acts as an intermediary between clients and backend services.
  - It routes traffic based on forwarding rule configuration and URL map rules.
  - **Benefits:**
    - Can perform security functions like SSL termination (especially with Envoy proxies in regional external ALBs).

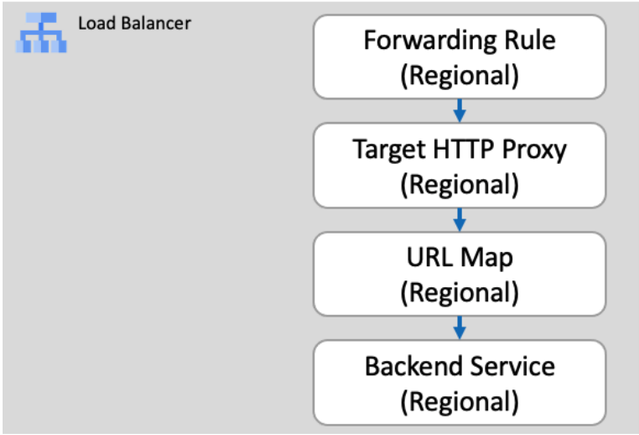




Google Cloud Load Balancing - Terminology

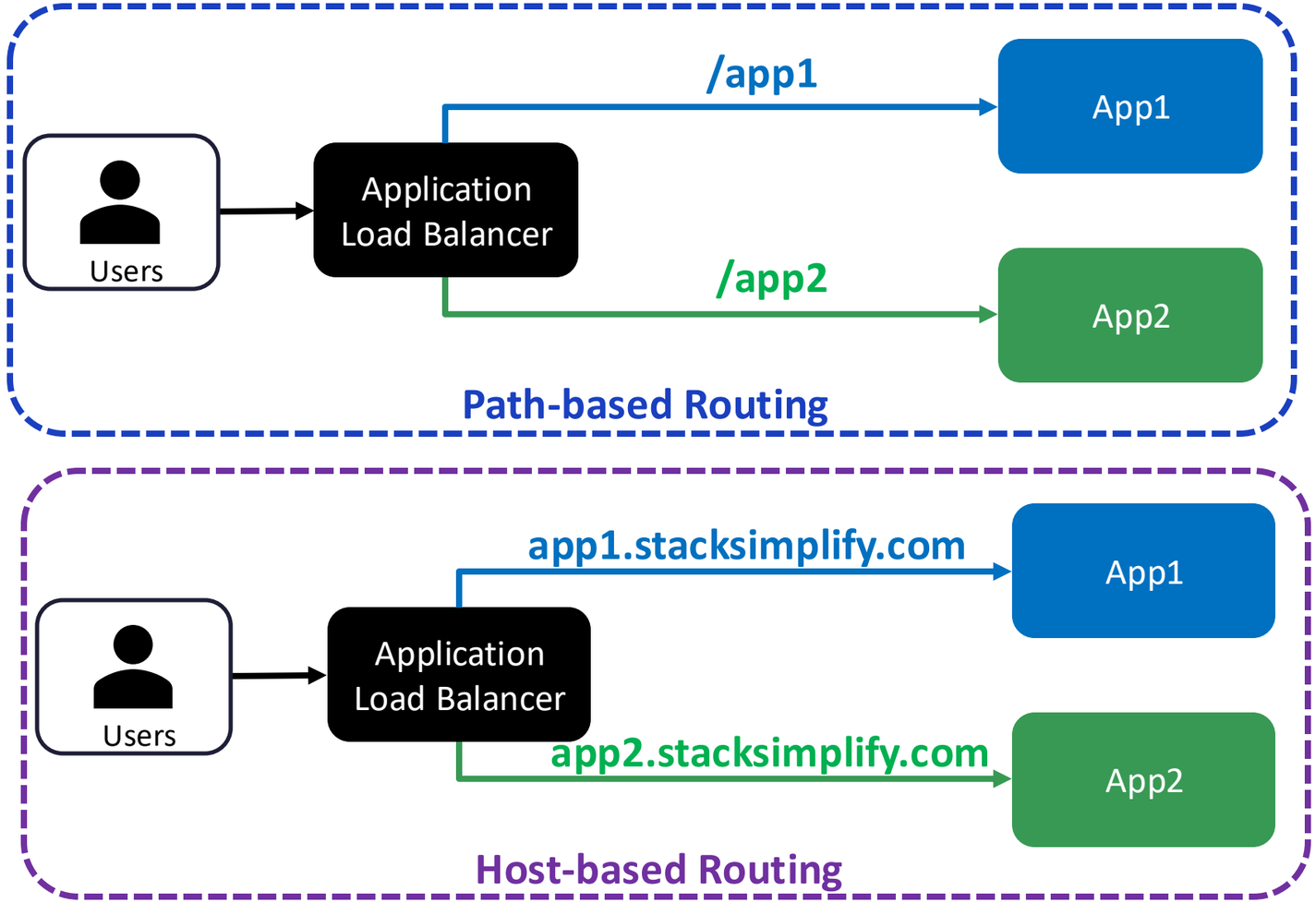
- **URL Map:**
  - Based on pre-defined rules (url path, host header, custom header) it determines the most suitable backend service to handle the request.
  - **Benefits:**
    - Enables intelligent traffic routing based on specific URL patterns.
      - Path based Routing
      - Domain based Routing
      - Custom Header Routing
- **Backend Service:**
  - Represents a group of backend instances (VMs, containers) that handle incoming requests routed by the load balancer.
  - It distributes traffic across healthy instances based on the chosen load balancing algorithm.
  - **Benefits:**
    - Provides high availability and scalability for your application by ensuring requests reach healthy instances.
    - Simplifies backend management by hiding individual instances from the load balancer configuration (you manage the backend service, not individual VMs).

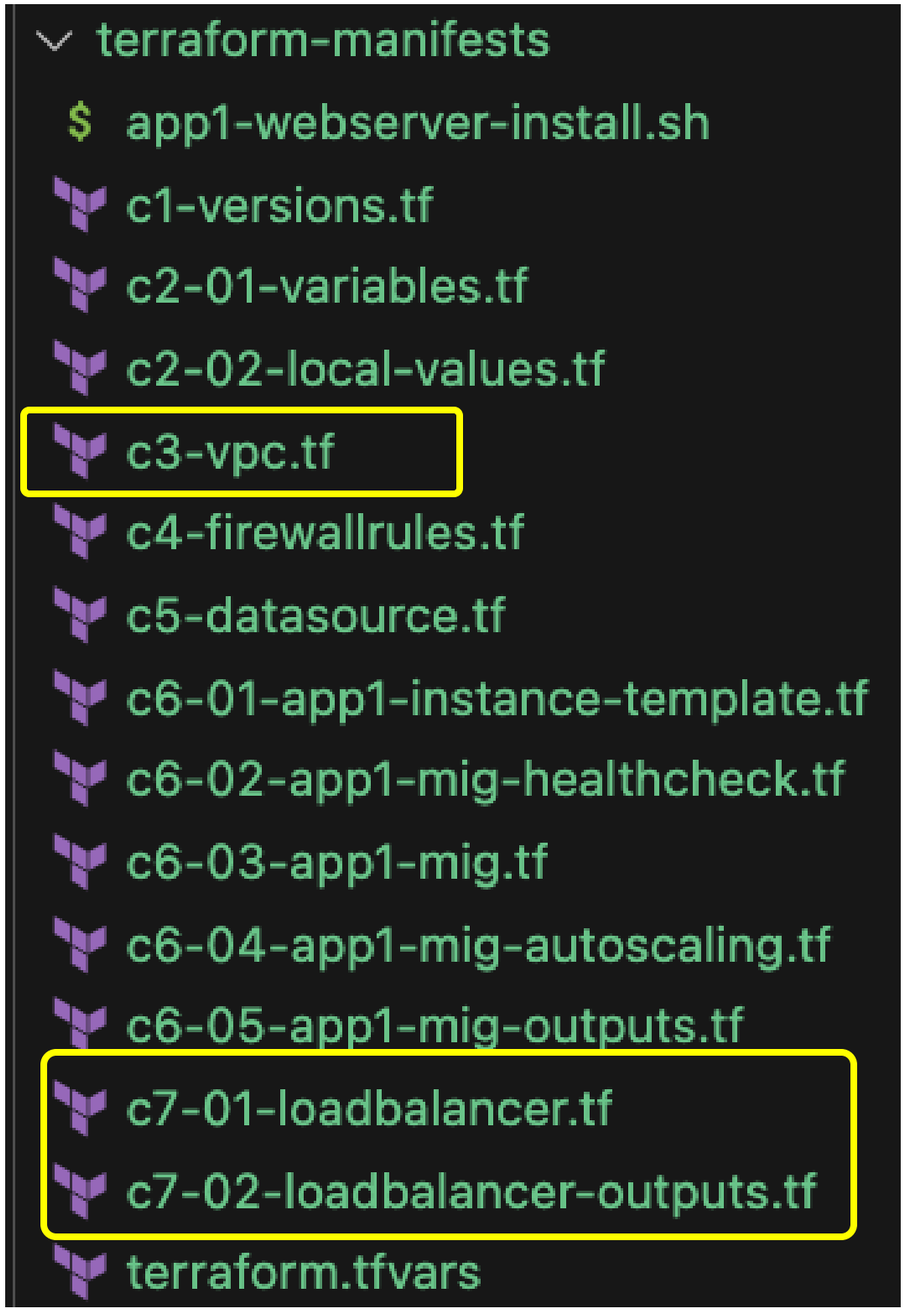




Google Cloud Load Balancing - Terminology

- **Routing Rules (Application LBs)**
  - **Path based Routing**
    - **App1:** stacksimplify.com/app1
    - **App2:** stacksimplify.com/app2
  - **Host based Routing**
    - **App1:** app1.stacksimplify.com
    - **App2:** app2.stacksimplify.com
  - **Rewrite the Requested URL**
    - **Host Rewrite:** app1.stacksimplify.com to app1.terraformguru.com
    - **Path Rewrite:** stacksimplify.com/app1 to stacksimplify.com/app1new
  - Add and remove request and response headers
  - Configure a URL redirect
  - Many many more things we can do





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Public IP + Regional Application Load Balancer HTTP

NC	No Changes from C1 to C6-05
C7-01	Create Regional Static IP Address and Regional Health check for Load Balancers
C7-01	Create Regional backend service and Regional URL Map
C7-01	Create Regional HTTP Proxy and Regional forwarding rule
C7-02	Terraform Outputs for Regional Application Load Balancer
C3	Create Regional Proxy-only subnet required for External Regional ALB

STACKSIMPLIFY

Demo-12

Terraform

GCP

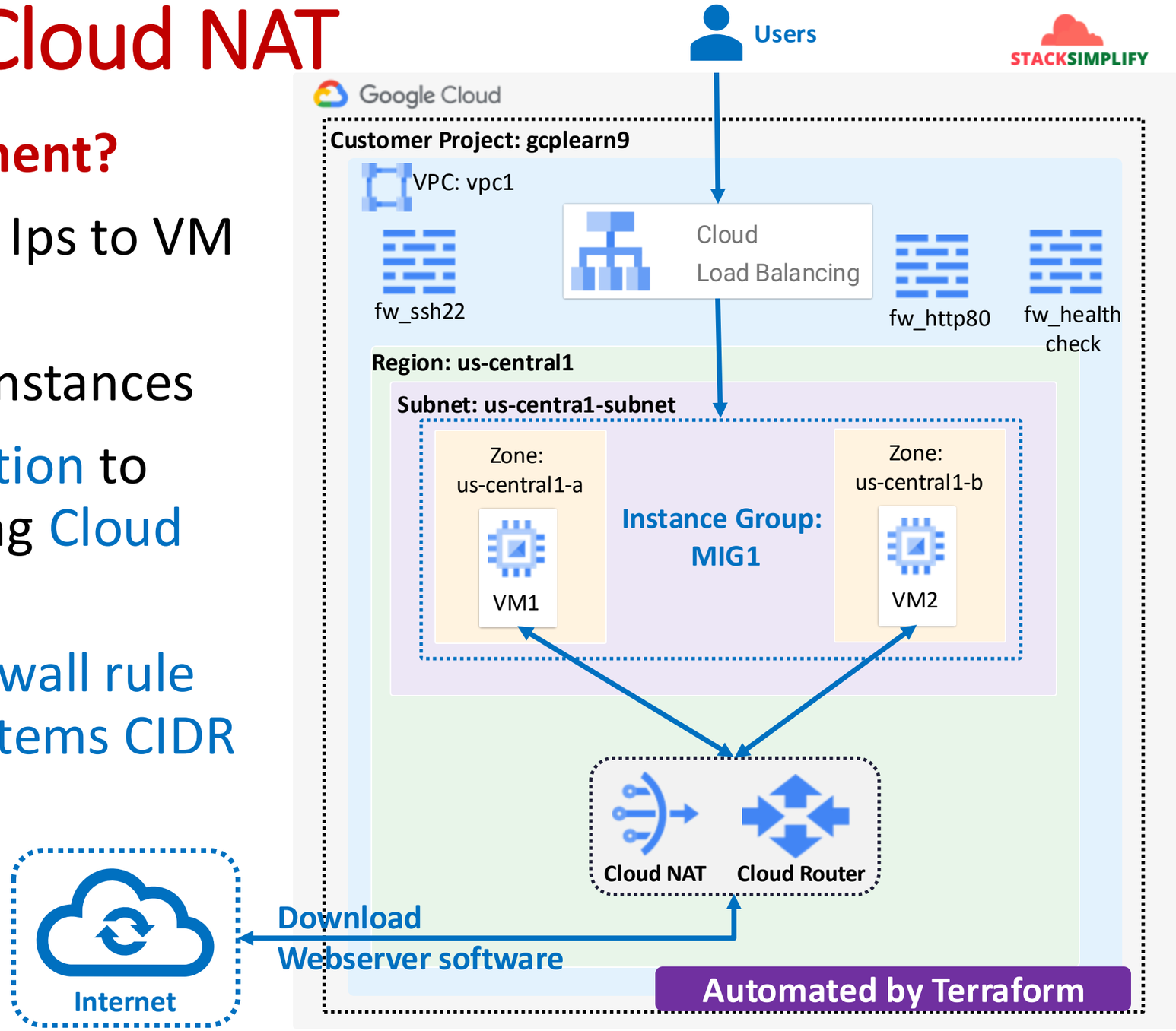
MIG Private IP

(Cloud NAT + Cloud Router)

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTP

GCP Cloud Router + Cloud NAT

- **What are we going to implement?**
- Remove External IPs or public Ips to VM Instances
- Have only private IPs for VM instances
- Enable outbound communication to internet for VM instances using Cloud Router and Cloud NAT
- Create allow health check firewall rule from Google Health check systems CIDR IP ranges



What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTP

NC	No Changes from C1 to C3, C5, C6-02 to C7-02
C4	Firewall rule to allow health check for private VMs from GCP Health Check CIDR ranges
C6-01	Update health check firewall rule tag, also update VMs to have only private Ips (Disable Public IP to VMs)
C8	Create Cloud Router and Cloud NAT for opening outbound communication to internet from GCP VMs

STACKSIMPLIFY

Demo-13

Terraform

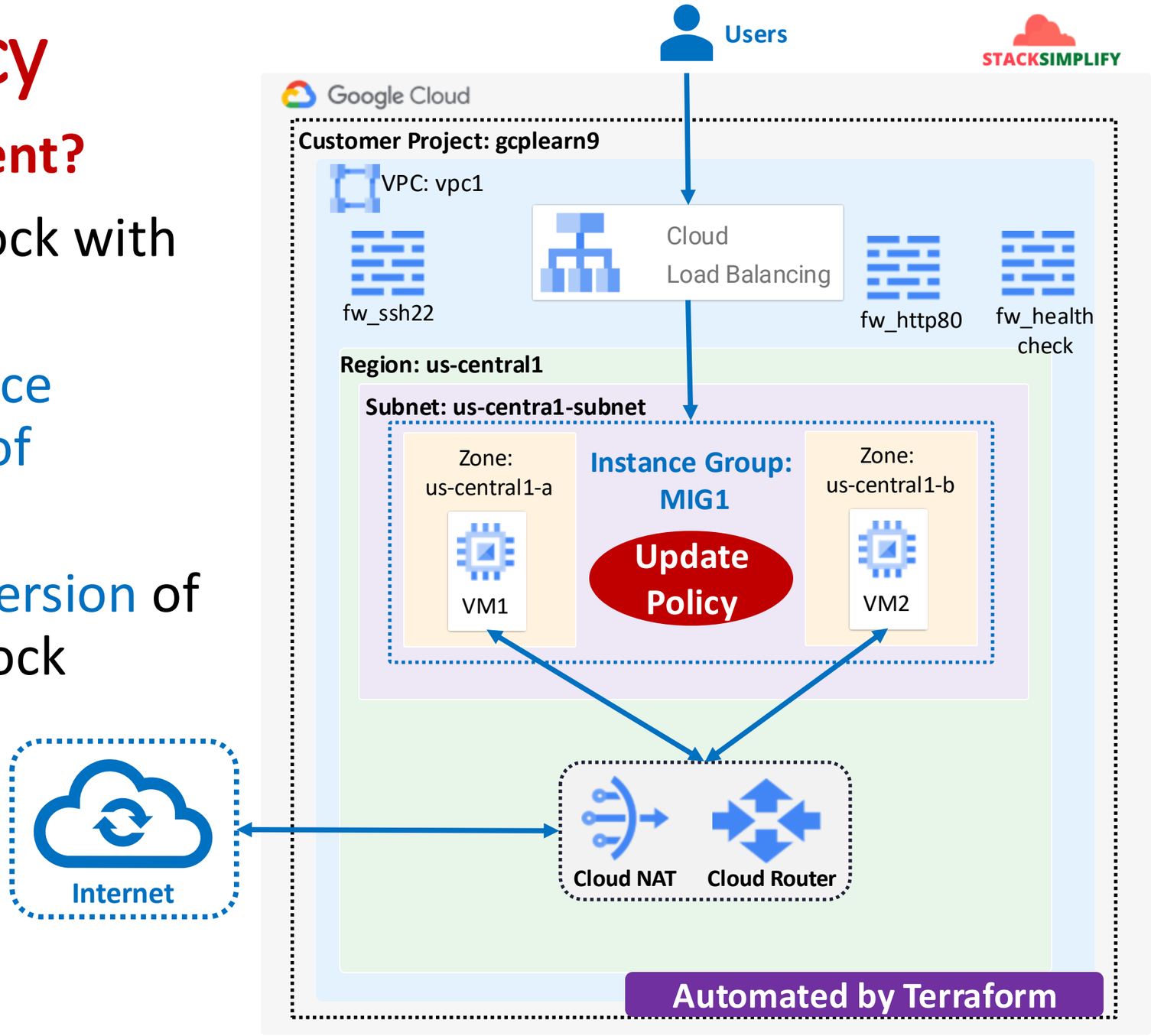
GCP

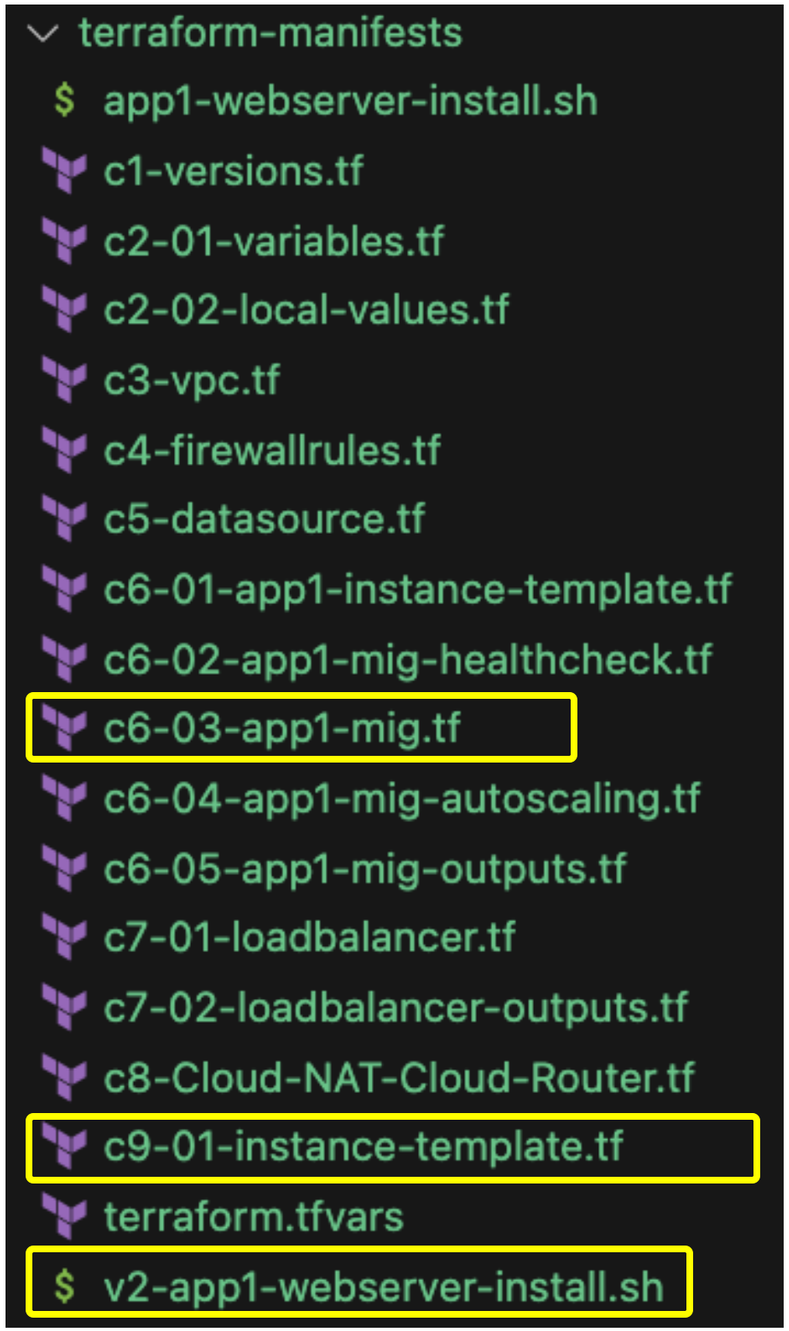
MIG Update Policy

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTP + MIG Update Policy

GCP MIG Update Policy

- **What are we going to implement?**
- **Step-01:** Add update_policy block with required settings in MIG
- **Step-02:** Create new VM instance template with new V2 version of startup-script
- **Step-03:** Update MIG with v2 version of instance template in version block





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTP + MIG Update Policy

NC	No Changes from C1 to C6-02, C6-04 to C8
v2	V2 version of application: v2-app1-webserver-install.sh
C9-01	Create new VM Instance template with v2-app1-webserver-install.sh
C6-03	Update version block in mig with v2 version of instance template
C6-03	Add update_policy block for MIG

STACKSIMPLIFY

Demo-14

Terraform

GCP Load Balancer

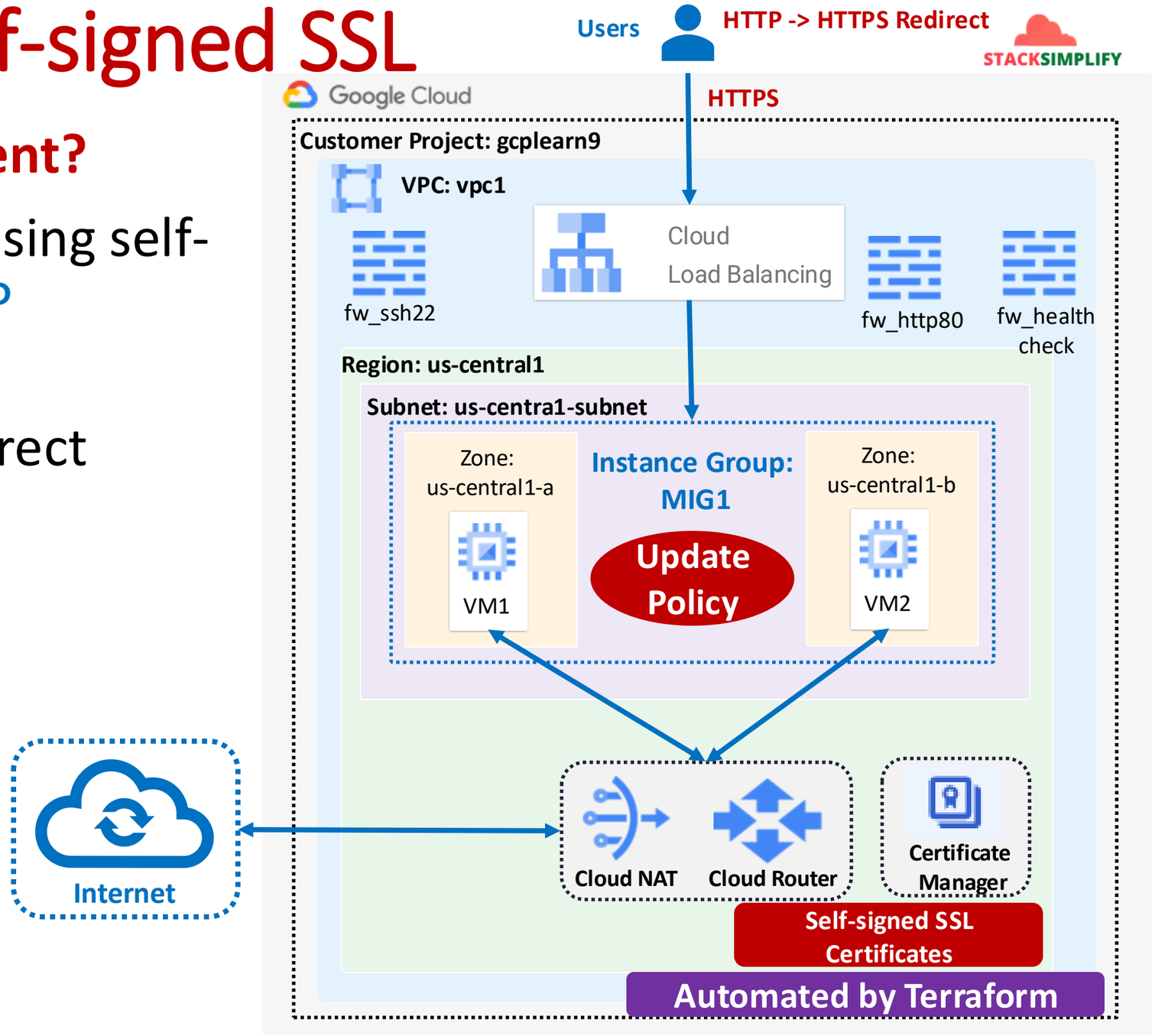
Self-Signed SSL using

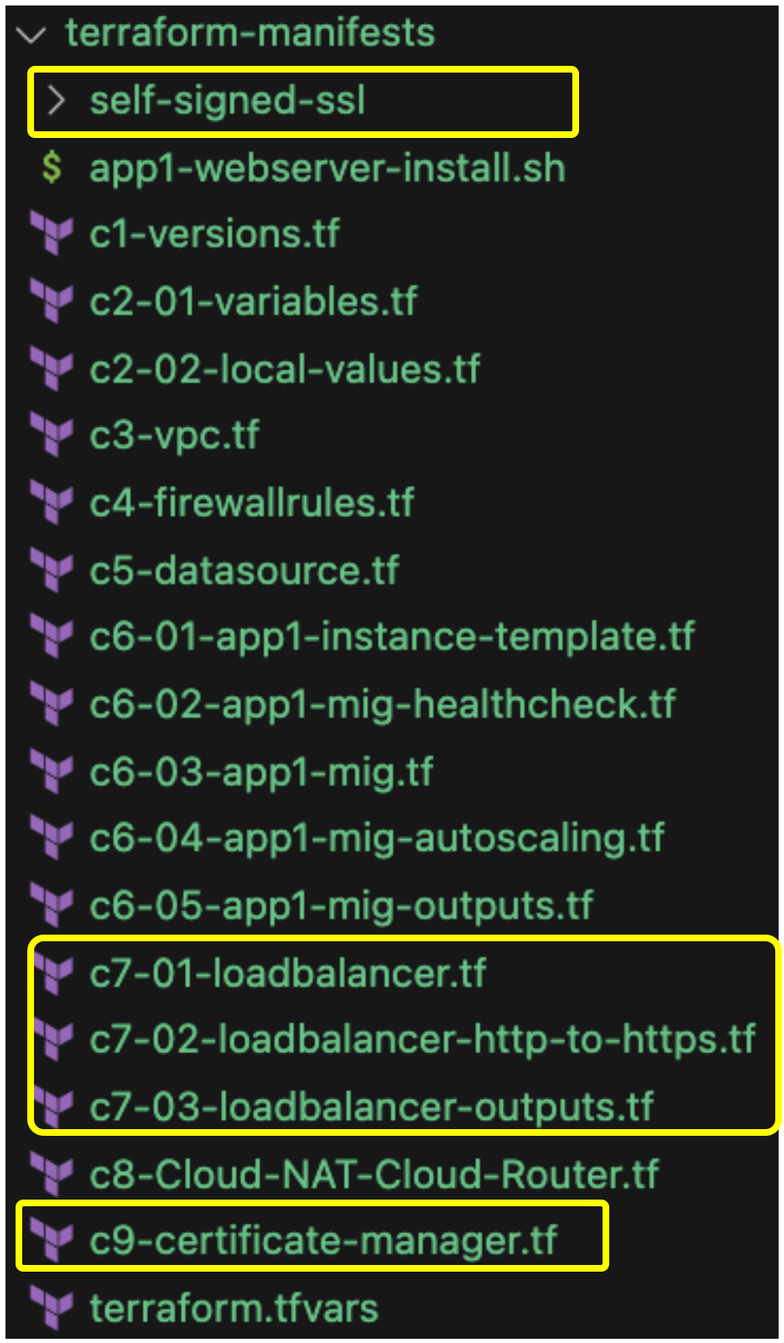
GCP Certificate Manager

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTPS + Self-signed SSL

GCP Load Balancer Self-signed SSL

- **What are we going to implement?**
- Create a HTTPS load balancer using self-signed SSL certificates with GCP certificate manager
- Implement HTTP to HTTPS redirect





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS + Self-signed SSL

NC	No Changes from C1 to C6-05, C8
SSL	Create self signed SSL private key and certificate
C9	Create self-signed SSL certificate object using certificate manager
C7-01	Comment HTTP proxy, create HTTPS proxy, Update regional forwarding rule with port 443
C7-02	Implement HTTP to HTTPS redirect with URL Map, HTTP Proxy, Forwarding rule
C7-03	Update HTTPS proxy output

STACKSIMPLIFY

Demo-15

Terraform

GCP Load Balancer

Cloud Domains + Cloud DNS

Demo

Google Cloud Networking

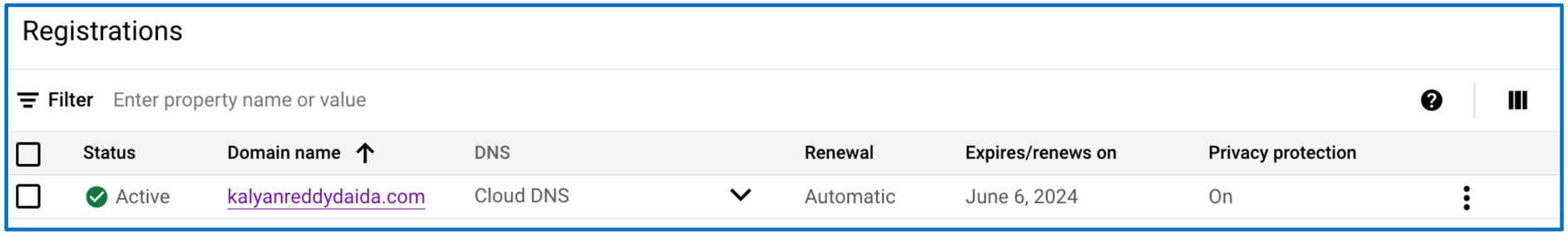
Cloud Domains

Google Cloud Domains

STACKSIMPLIFY

Cloud Domains

- **Cloud Domains:** Primarily used for Domain Registration and management
- **Key Benefits**
  - Register new domains
  - Bills for purchasing/renewing domains will use the same Cloud Billing account
  - Automatic renewal of registered domains
  - Let's you manage domain registrations per project, not per user
  - Supports DNSSEC which protects your domains from spoofing and cache poisoning attacks
  - Tightly integrated with Google Cloud DNS (Domain Name system) for creating and managing DNS records



Google Cloud Domains

STACKSIMPLIFY

Cloud Domains

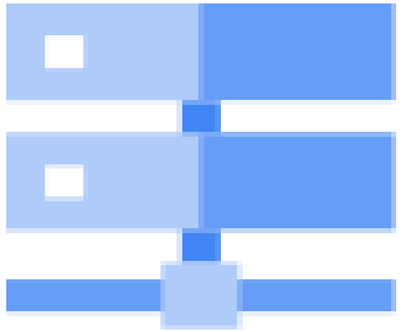
- **Important Note:** From September 7, 2023 onwards Squarespace acquired all domain registrations and related customer accounts from Google Domains
- Cloud Domains uses Google Domains as underlying domain registrar
- Now after squarespace acquisition, Cloud Domains uses Squarespace Domains as underlying registrar
- **How does this impact Cloud Domains in Google Cloud ?**
  - **Complete Faq:** https://cloud.google.com/domains/docs/faq
  - In a shorter note, google supports Cloud Domains even after this acquisition
  - **Following features in Cloud Domain will work as-is**
    - Searching and registering new domains
    - Renewing existing domains
    - Update your contact details and DNS settings
    - Transfer a registered domain to another registrar (GoDaddy, Namecheap, AWS Route53)
    - Transfer domain from another registrar to Cloud Domain - NOT POSSIBLE

Additional Reference: https://cloud.google.com/domains/docs/deprecations/feature-deprecations

Demo

Google Cloud Networking

Cloud DNS



Google Cloud DNS

- **Cloud DNS:** High-performance, resilient, global Domain Name System (DNS) service
- Cloud DNS translates requests for domain names like stacksimplify.com into IP addresses like 13.224.249.31

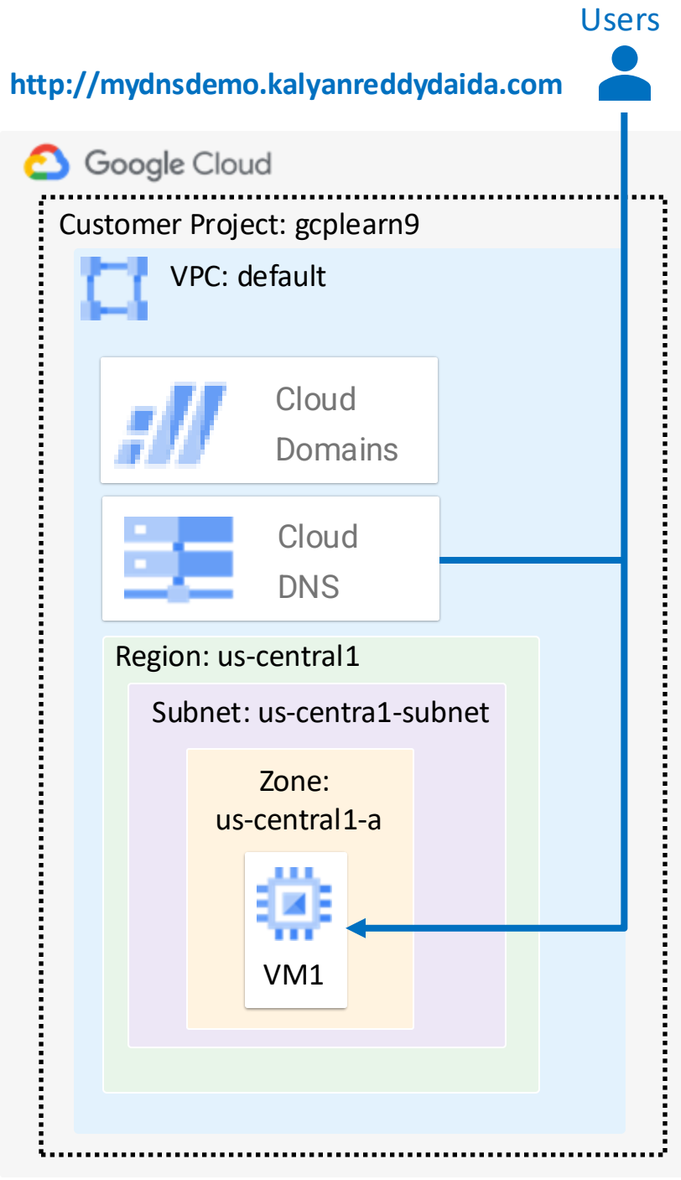
```
dkalyanreddy@cloudshell:~ (gcplearn9)$ nslookup stacksimplify.com
Server:         169.254.169.254
Address:        169.254.169.254#53

Non-authoritative answer:
Name:   stacksimplify.com
Address: 13.224.249.31
Name:   stacksimplify.com
Address: 13.224.249.62
Name:   stacksimplify.com
Address: 13.224.249.81
Name:   stacksimplify.com
Address: 13.224.249.73
```

Google Cloud DNS - Terminology

- **DNS Zones:** It is a container of DNS records for the same DNS name suffix (Example suffix: stacksimplify.com)
  - **Public Zone**
    - Visible to the internet
    - Primarily used for DNS management of your internet facing applications
    - DNSSEC (DNS Security) protects your domains from spoofing and cache poisoning attacks

Zone name ↑	DNS name	DNSSEC	Description	Zone type
kalyanreddydaida-com	kalyanreddydaida.com.	On	DNS zone for domain: kalyanreddydaida.com	Public

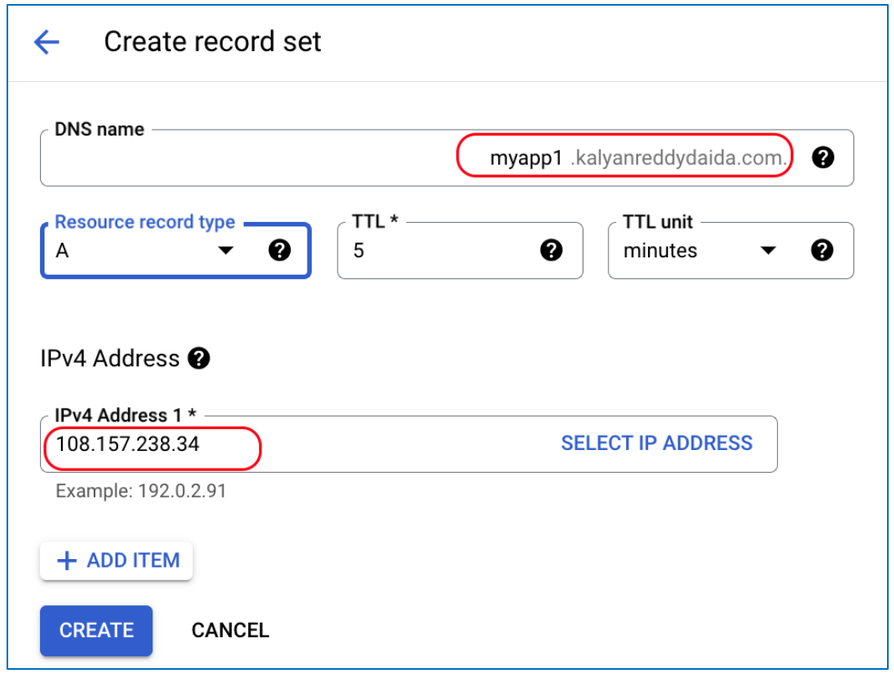
  - **Private Zone**
    - Contains DNS records that are only visible internally within your Google Cloud networks
    - Easy to manage internal DNS solution for all our internal needs (For Internal applications, VM Instances etc)



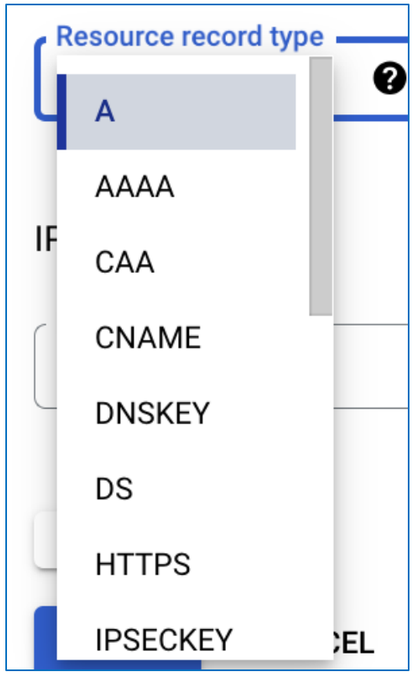
Google Cloud DNS - Terminology

- **Record Sets:** actual DNS records
  - DNS Name to IP Address mapping
  - We have different record types, but primarily we use Address record (A) type which maps hostnames to IPv4 address

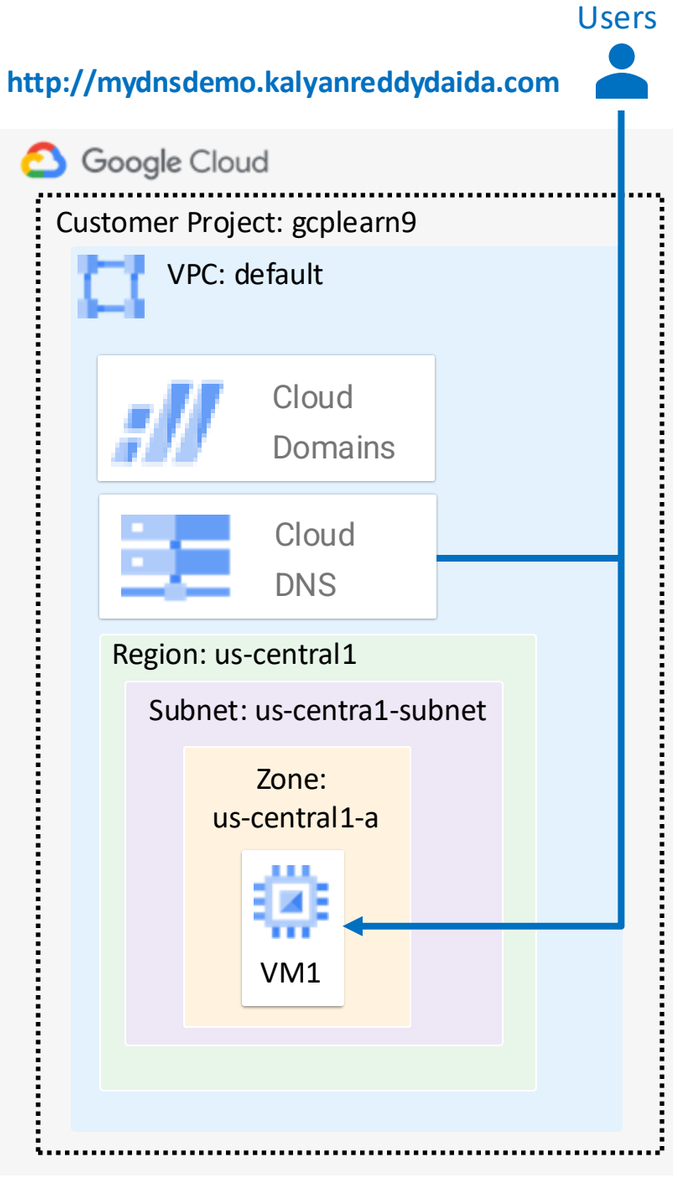
DNS Record Set



DNS Record Type

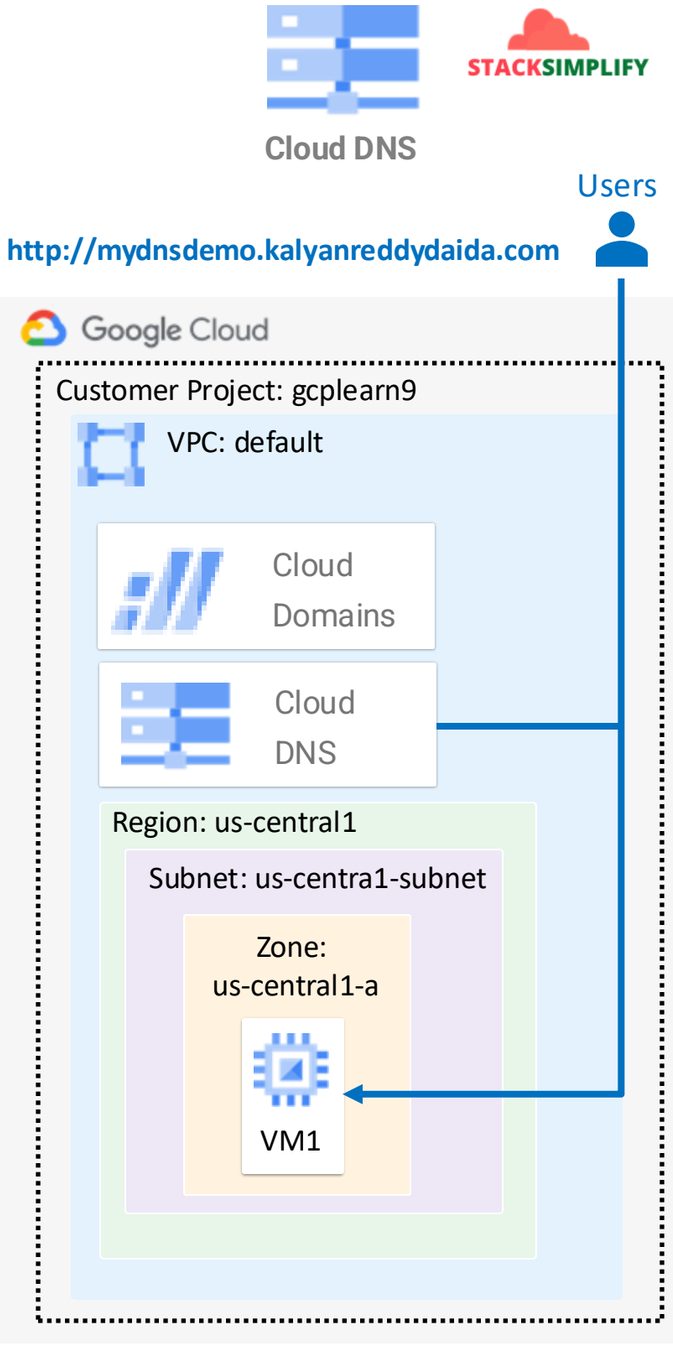


https://cloud.google.com/dns/docs/records-overview#supported_dns_record_types



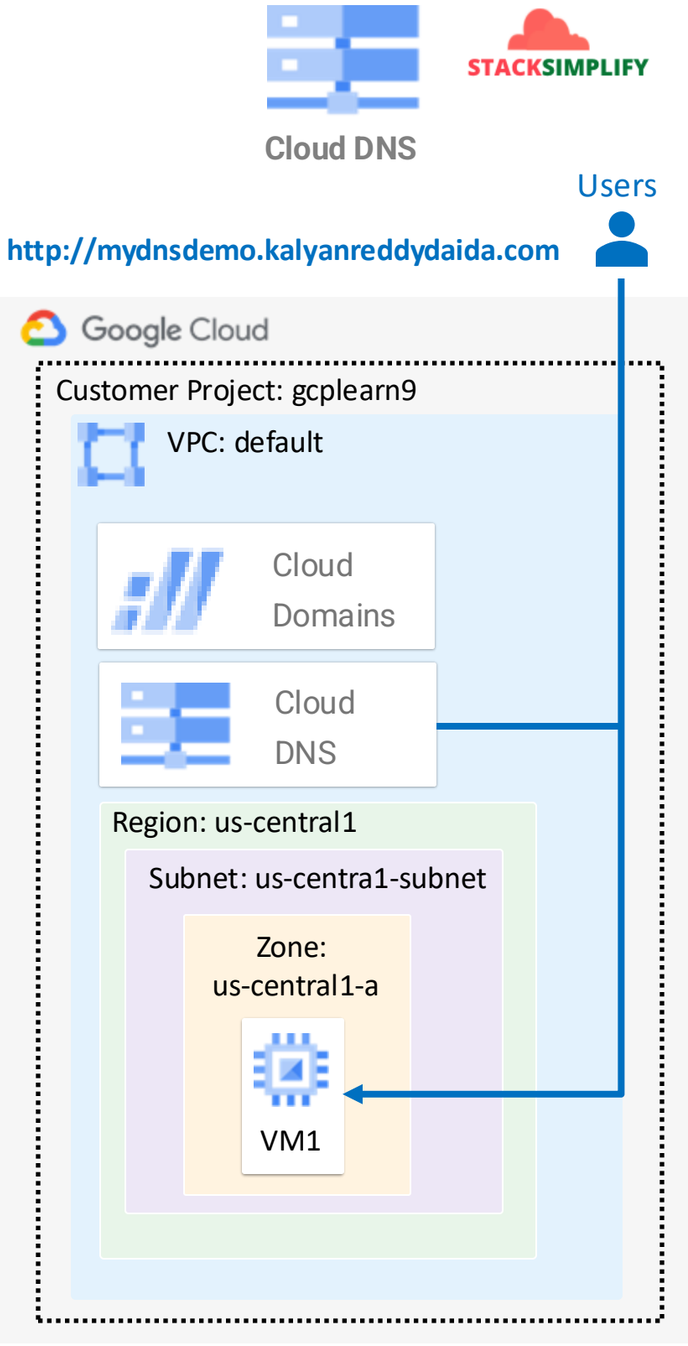
Google Cloud DNS - Features

- **Integration with Cloud IAM**
  - Provides secure domains management with full control and visibility for domain resources
- **Integration with Cloud Logging**
  - Private DNS logs a record for every DNS query in Cloud Logging
  - We can view and export logs to supported destinations
- **Fast anycast Nameservers**
  - **Anycast:** Uses the nearest nameserver to users for DNS lookup and resolution
  - Provides very high availability across globe and low latency
- **DNS registration and management**
  - Cloud DNS is tightly integrated with Cloud Domains which takes care of DNS registration and management



Google Cloud DNS - Features

- **Container-native Cloud DNS**
  - Natively integrated with Google Kubernetes Engine (GKE)
  - Provides in-cluster Service DNS resolution
  - Provides high-throughput, scalable DNS resolution for every GKE node
- **DNS Peering**
  - Sharing DNS data
  - All or a portion of DNS namespace can be configured to be sent from one network to another
  - Will respect all DNS configurations defined in peered network
- **DNS Forwarding**
  - Primarily used for hybrid-cloud architecture
  - Forward DNS queries from Cloud DNS to on-premise DNS servers where actual DNS resolution takes place



STACKSIMPLIFY

Pre-requisite:
Registered Domain is required to implement this demo

Demo-16

Terraform

GCP Load Balancer

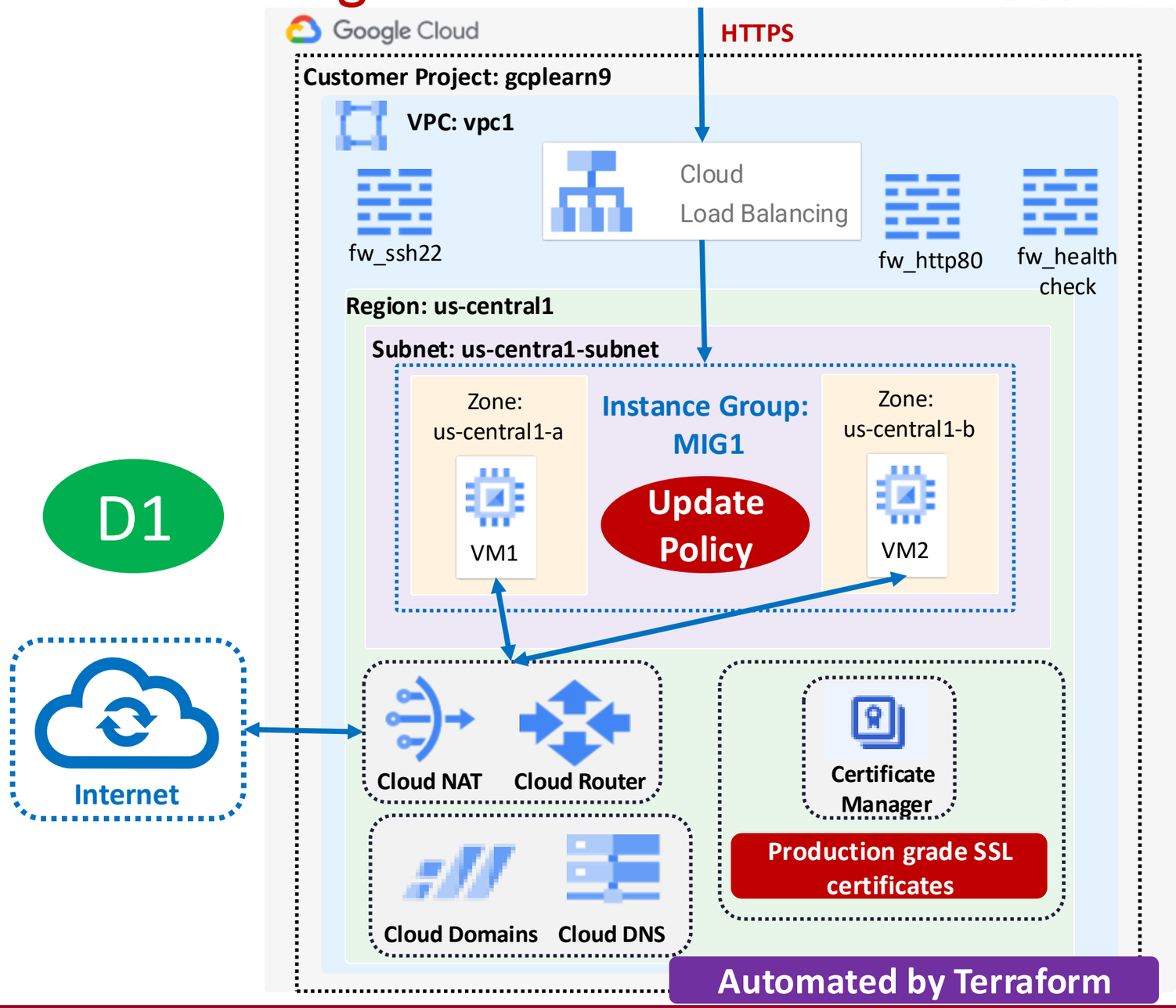
Cloud Domains + Cloud DNS

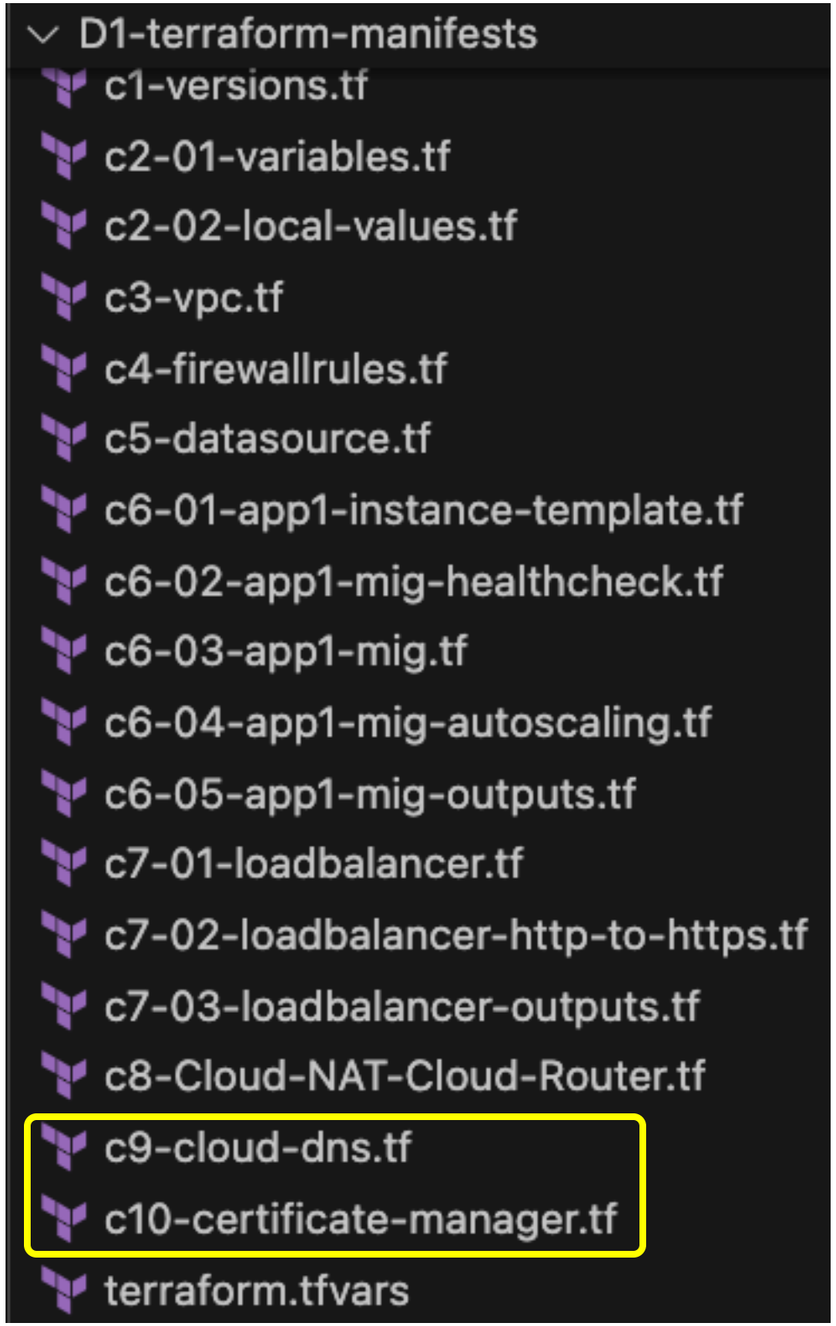
Production grade SSL Certificates

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS + Cloud DNS + Cloud Domians

GCP Cloud DNS + Cloud Domains as Domain Registrar

- **What are we going to implement?**
- **Pre-requisite:**
  - Registered domain using Cloud Domains
  - Cloud DNS zone pre-created and ready to use
- Create production grade SSL certificate using Certificate Manager with DNS Authorization
- Create a HTTPS load balancer
- Create DNS record in Cloud DNS for myapp1 application





What are we going to learn?

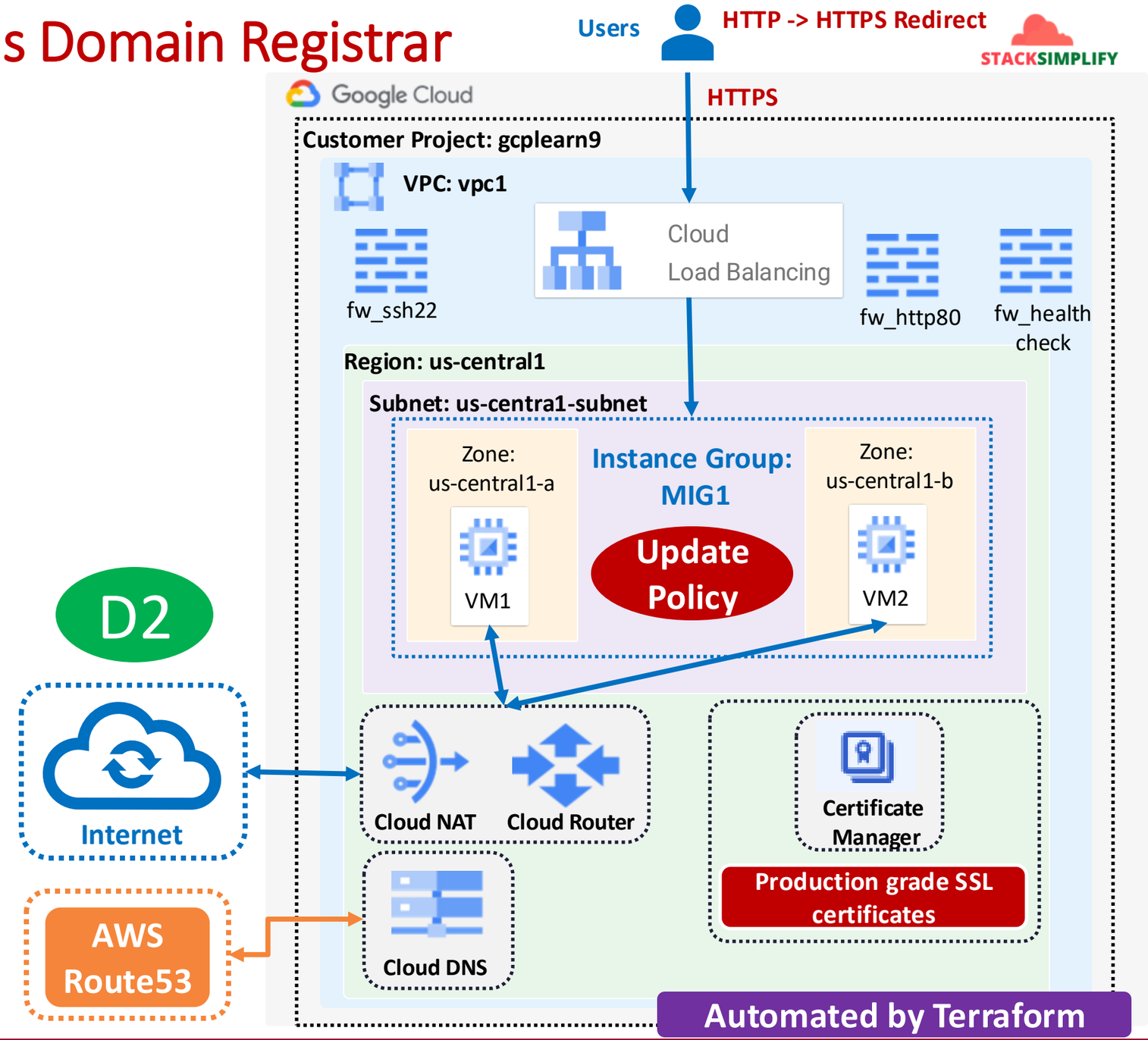
STACKSIMPLIFY

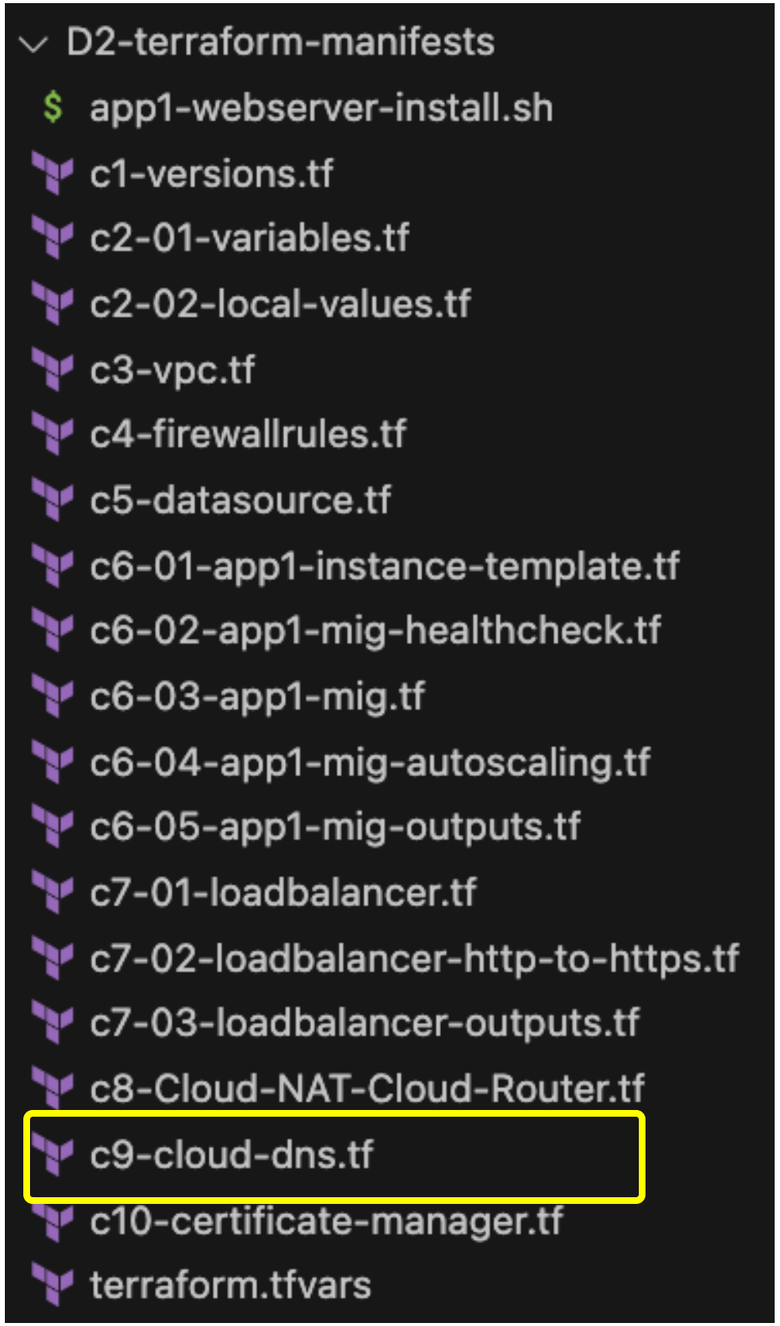
Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS + Cloud Domains + Cloud DNS

NC	No Changes from C1 to C8
C9	Create a Cloud DNS record set
C10	Create SSL certificate using certificate manager
C10	Create certificate manager DNS authorization
C10	Create Cloud DNS record set for DNS authorization

GCP Cloud DNS + AWS Route53 as Domain Registrar

- **What are we going to implement?**
- **Pre-requisite:**
  - Registered domain using any domain provider like GoDaddy, namecheap, AWS Route53
- Create Cloud DNS zone
- Update AWS Route53 with Google Cloud DNS Nameservers
- Create production grade SSL certificate using Certificate Manager with DNS Authorization
- Create a HTTPS load balancer
- Create DNS record in Cloud DNS for myapp1 application





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS + Cloud Domains + Cloud DNS

NC	No Changes from C1 to C8
C9	Update locals block with DNS Name and Cloud DNS Zone

STACKSIMPLIFY

Demo-17

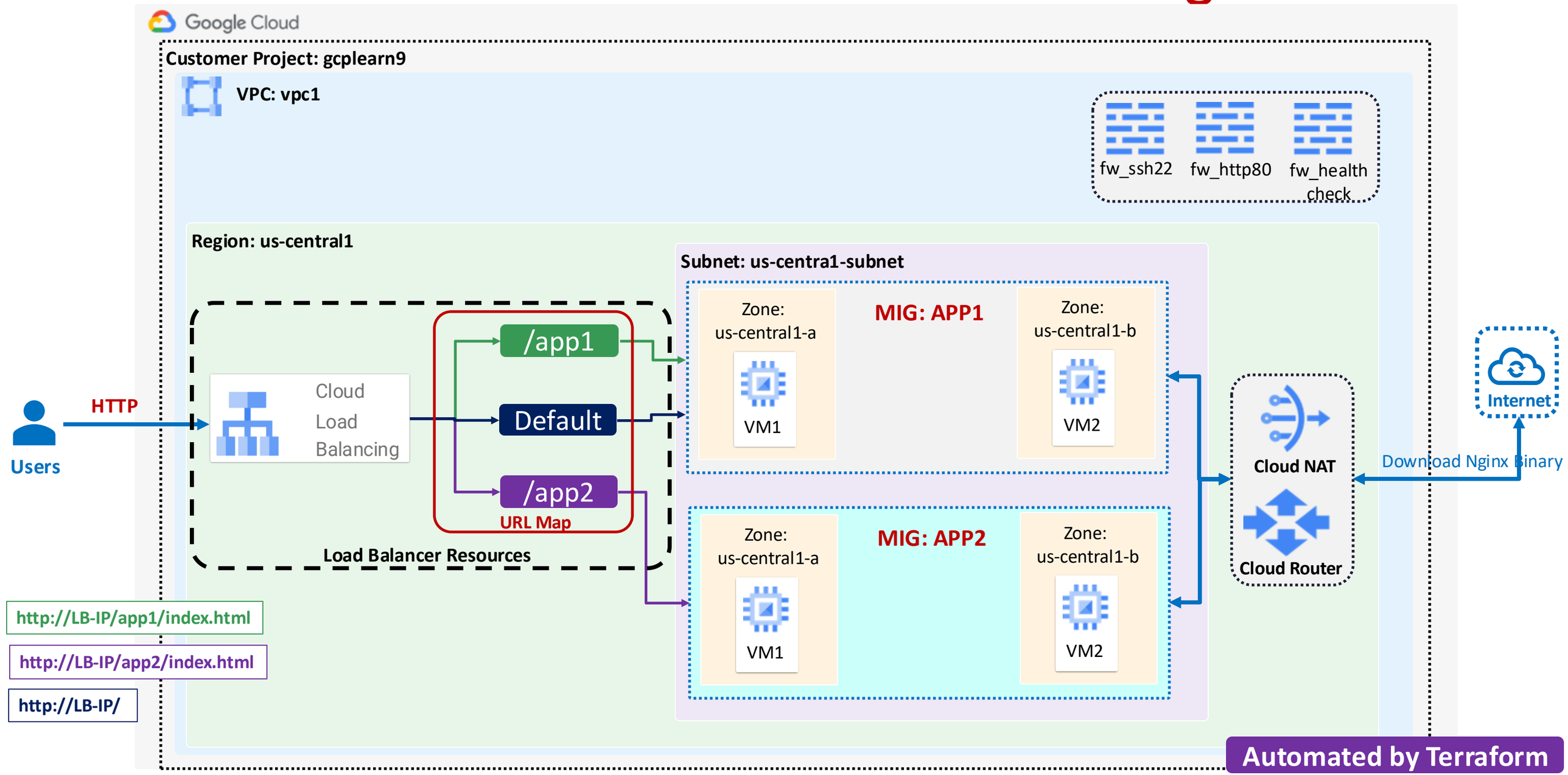
Terraform

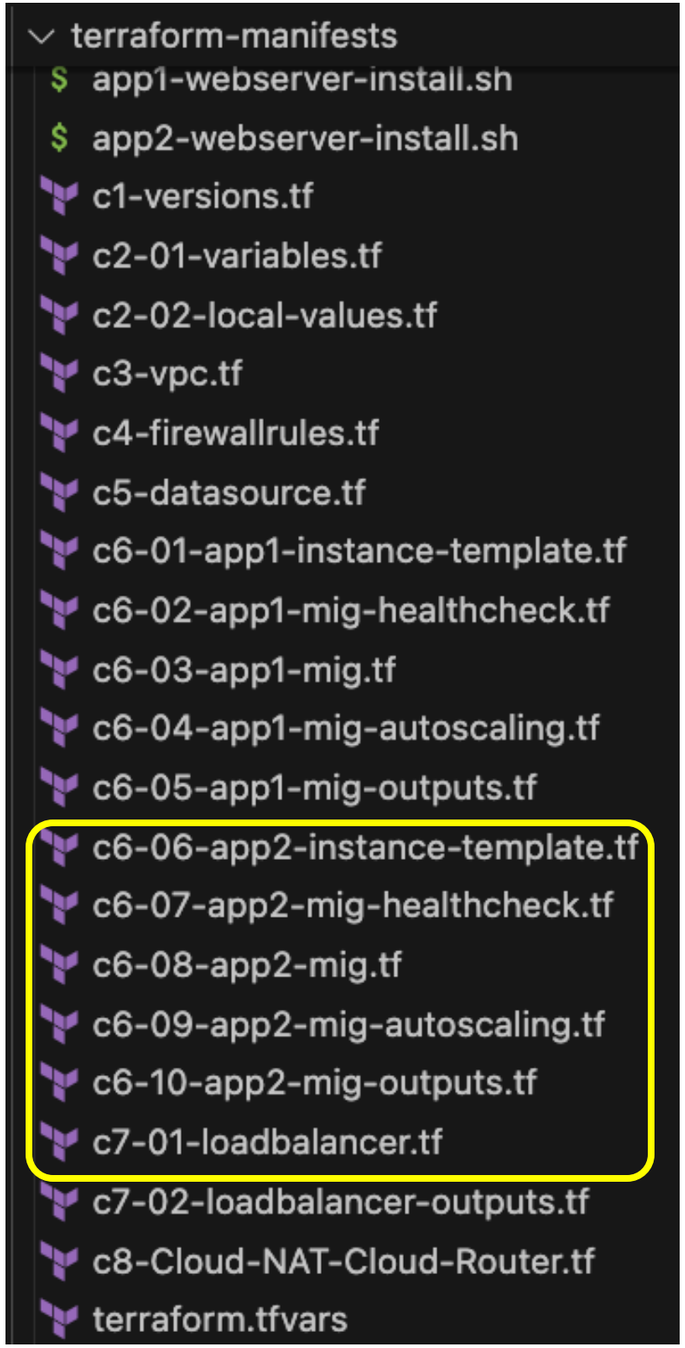
GCP Load Balancer

HTTP Context Path Routing

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTP + Context Path Routing

GCP Load Balancer + Context Path Routing





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTP + Context Path Routing

NC	No Changes from C1 to C6-05, C7-02, C8
APP2	Create App2 from C6-06 to C6-10
C7-01	Create App2 Backend Service
C7-01	Create URL Map with context path-based routing

STACKSIMPLIFY

Demo-18

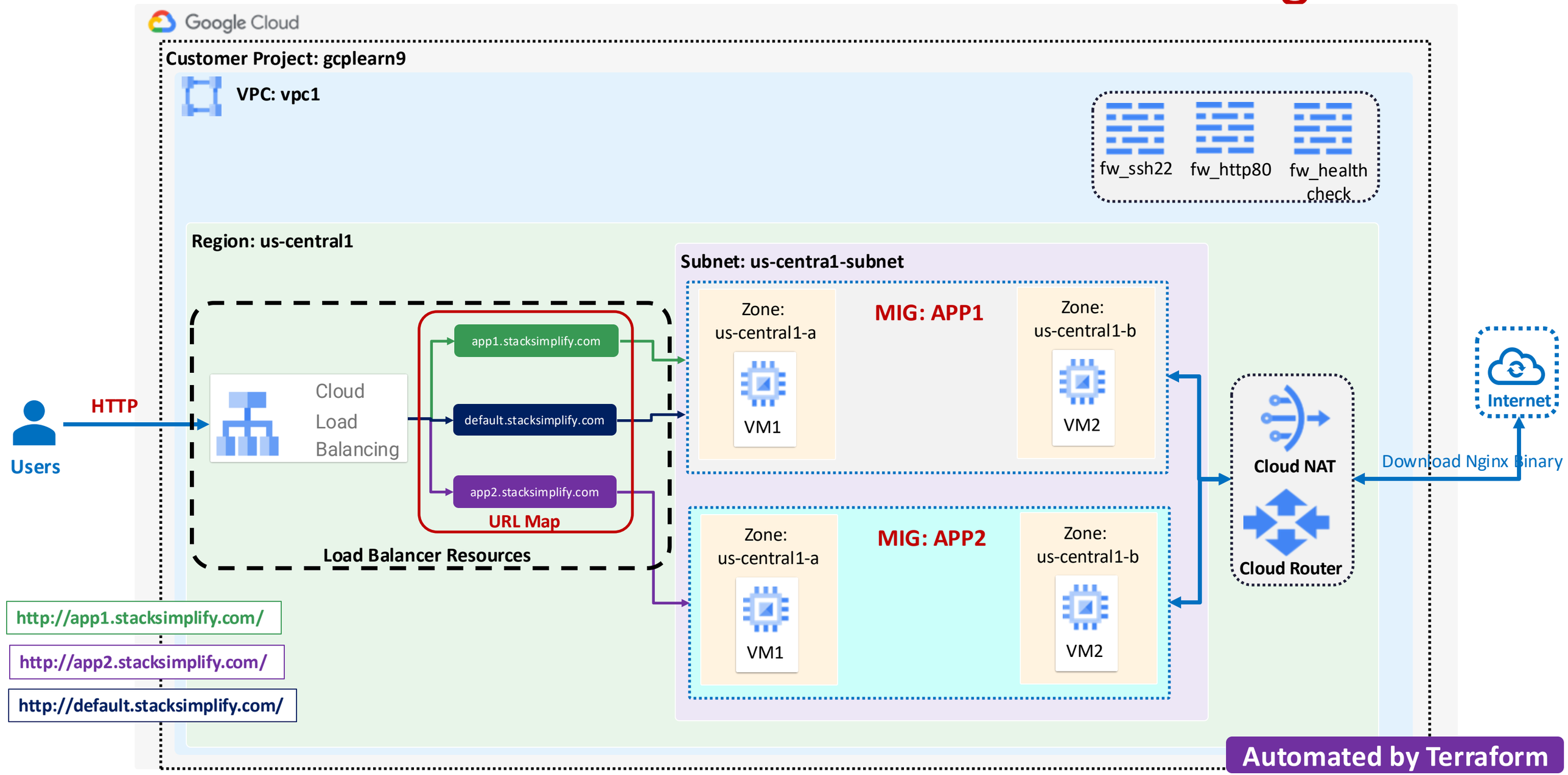
Terraform

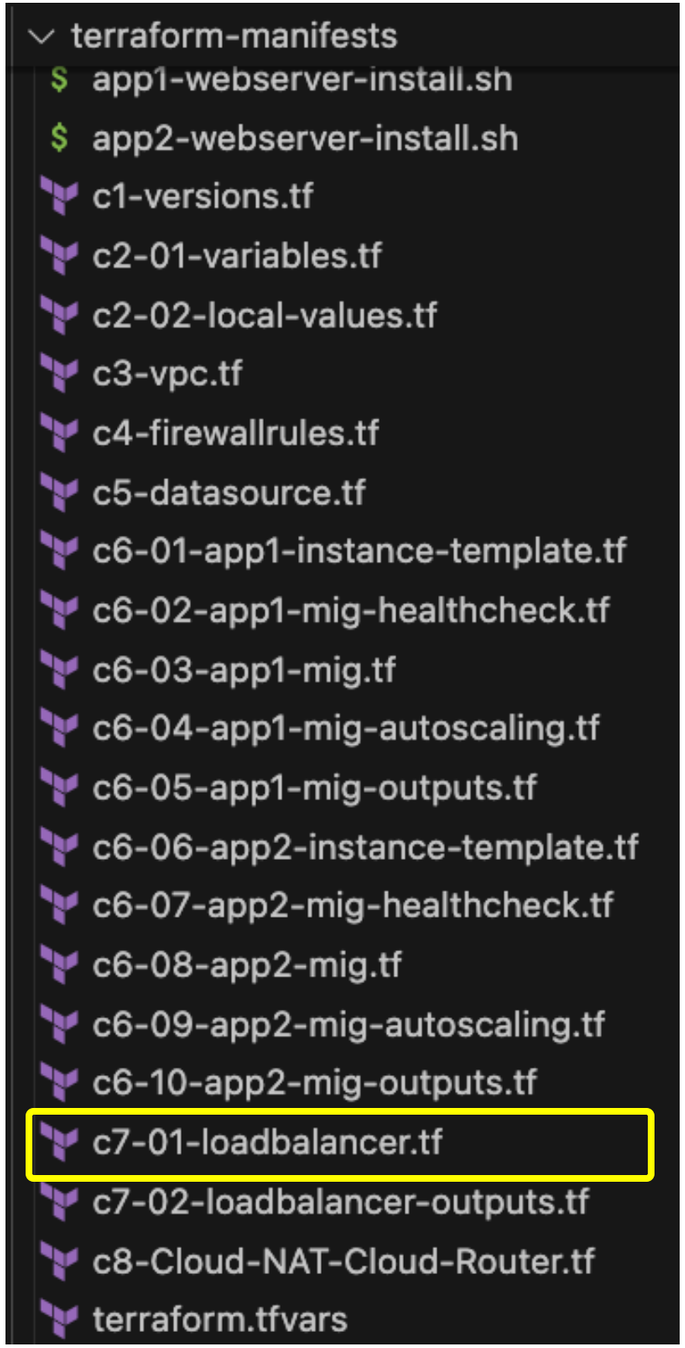
GCP Load Balancer

Domain Name based Routing

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTP + Domain Name based Routing

GCP Load Balancer + Domain Name based Routing





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTP + Domain Name based Routing

NC	No Changes from C1 to C6-10, C7-02, C8
C7-01	Create URL Map for domain name-based routing

STACKSIMPLIFY

Demo-19

Terraform

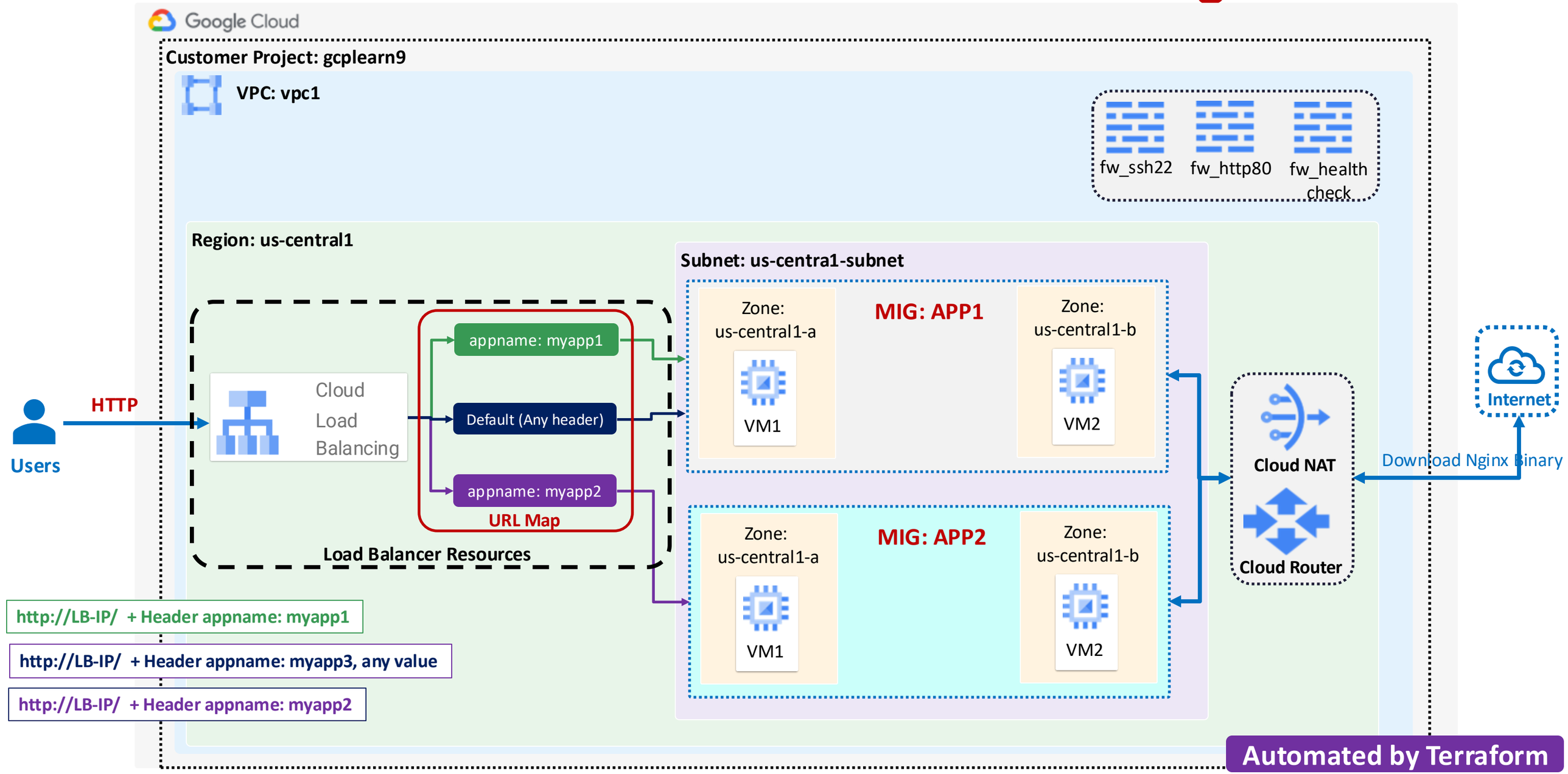
GCP Load Balancer

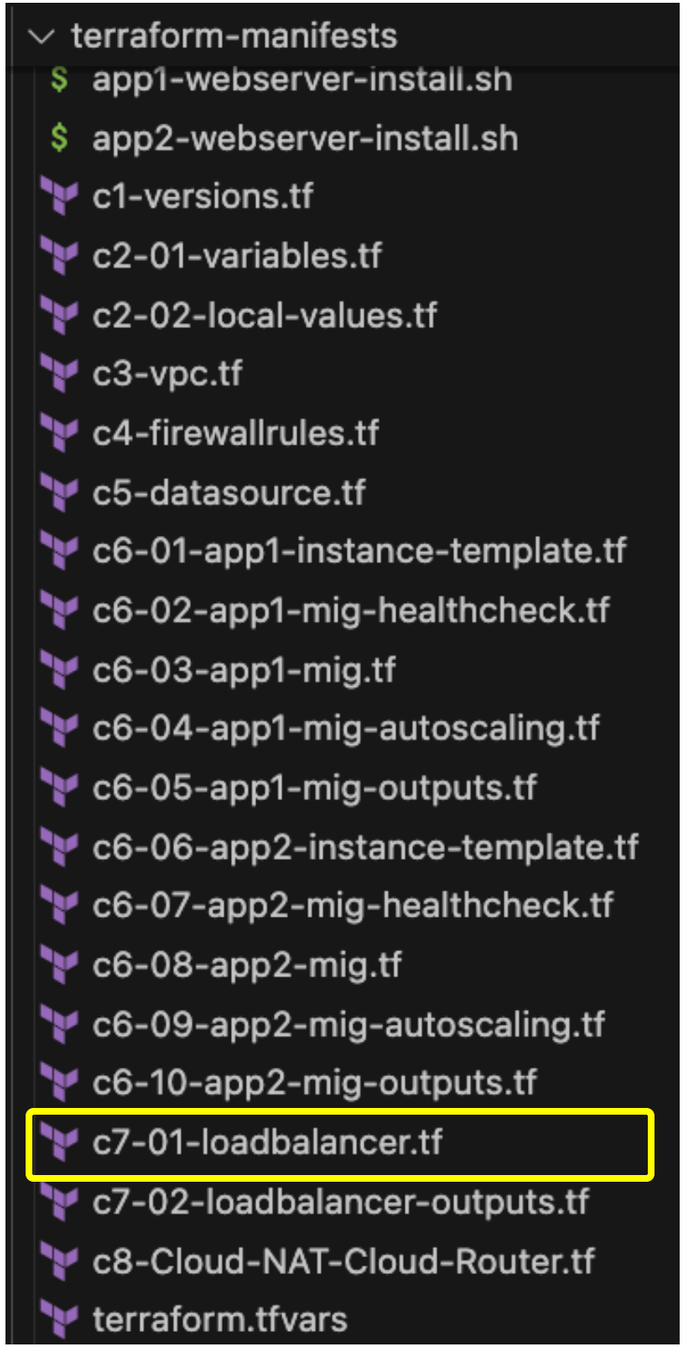
Header based Routing

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTP + Header based Routing

GCP Load Balancer + Header based Routing

STACKSIMPLIFY





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTP + Header based Routing

NC	No Changes from C1 to C6-10, C7-02, C8
C7-01	Create URL Map for Header based routing

STACKSIMPLIFY

Demo-20

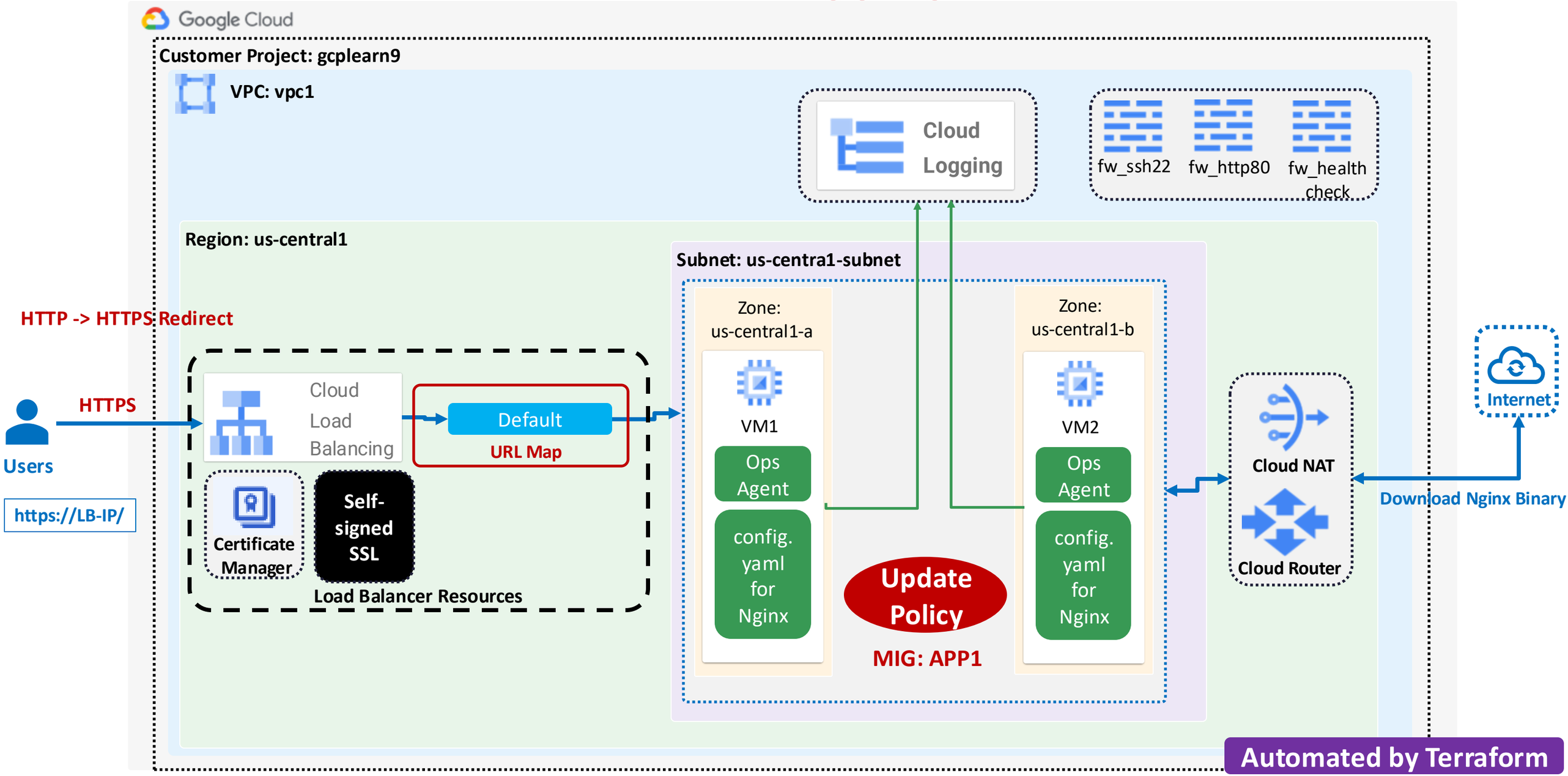
Terraform

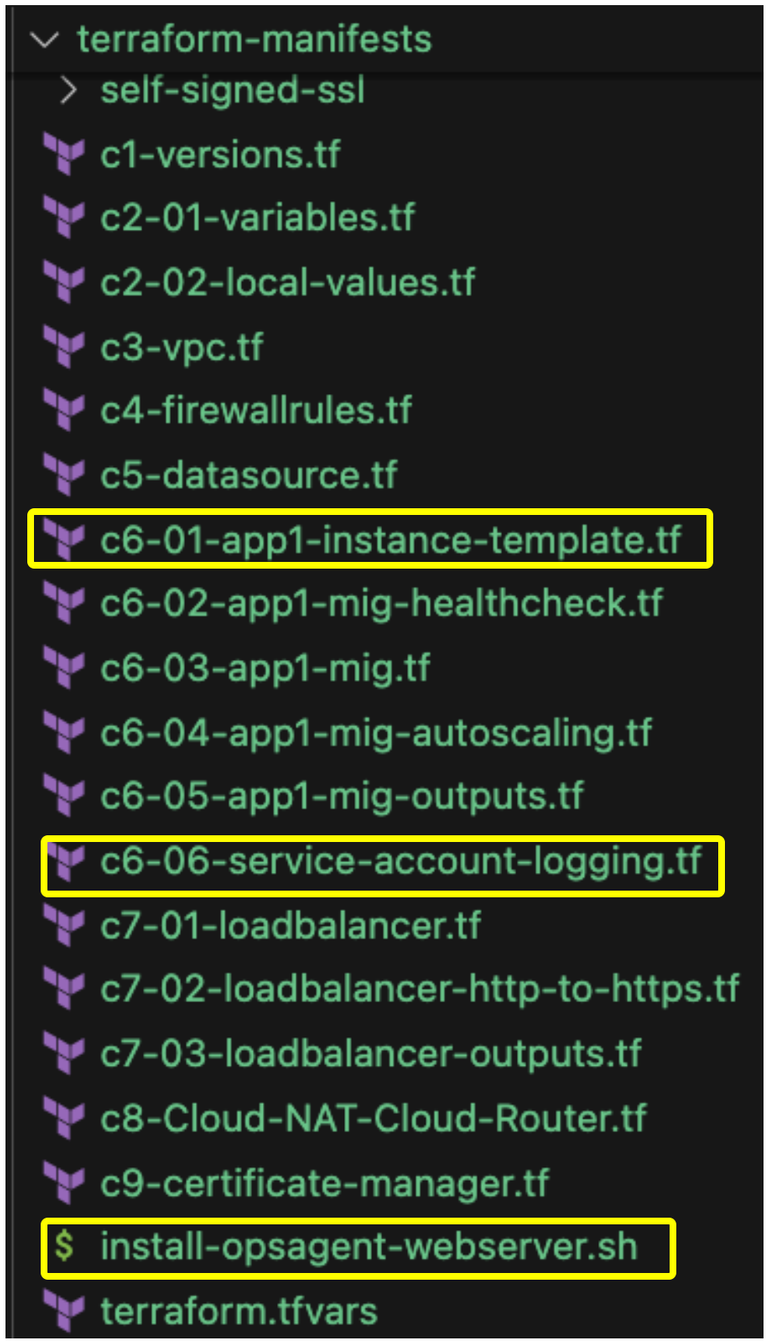
Application Logging

Cloud Logging

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTPS (Self-signed SSL) + LOGGING

GCP Cloud Logging





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Self-signed SSL) + Logging

NC	No Changes from C1 to C5, C6-02 to C6-05, C7-01 to C9
Ops Agent	Install Ops Agent install-opsagent-webserver.sh
C6-06	Create Service Account for Logging with Log writer and metric writer roles
C6-01	Update instance template with service_account block

Demo-21

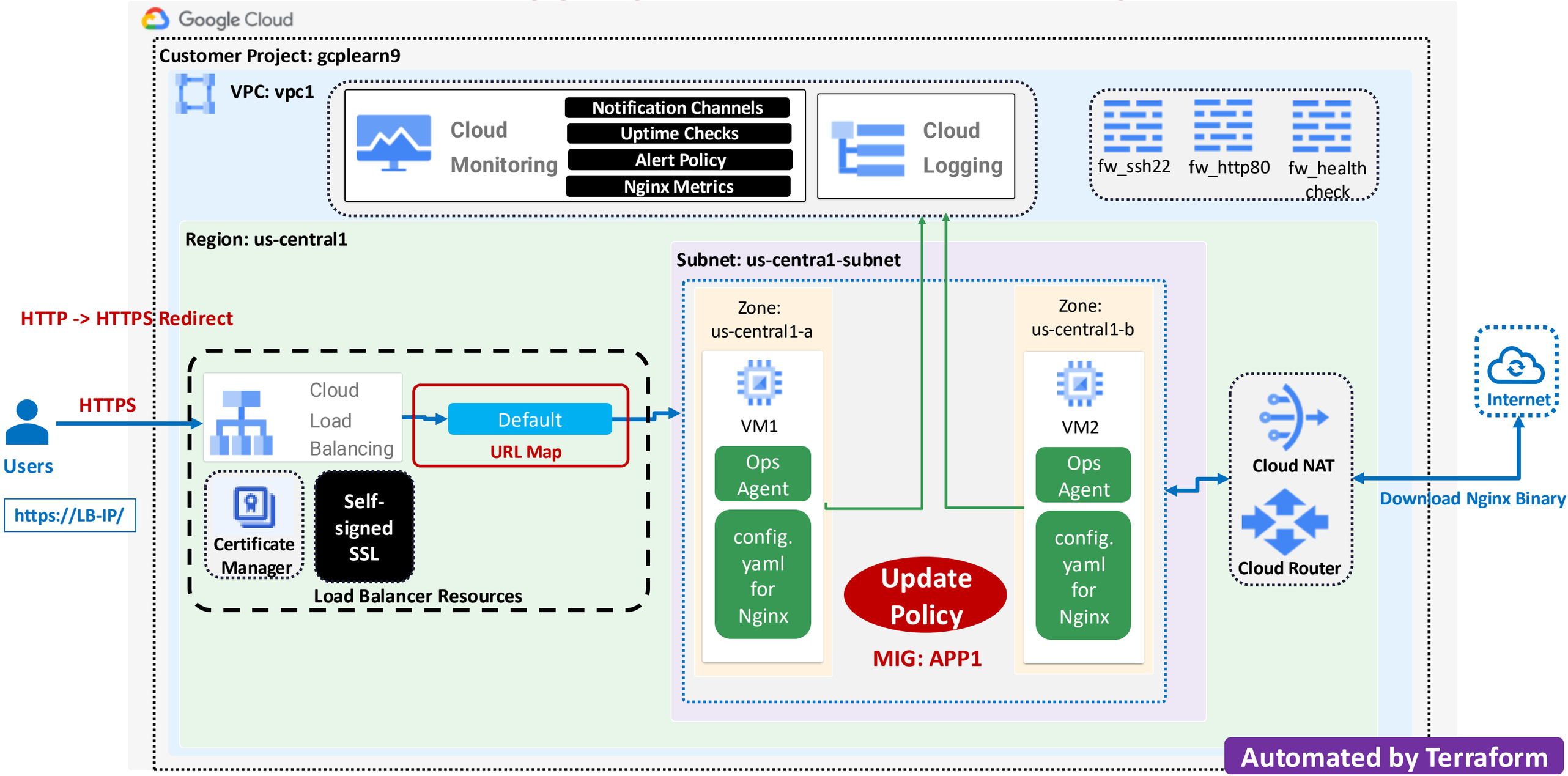
Terraform

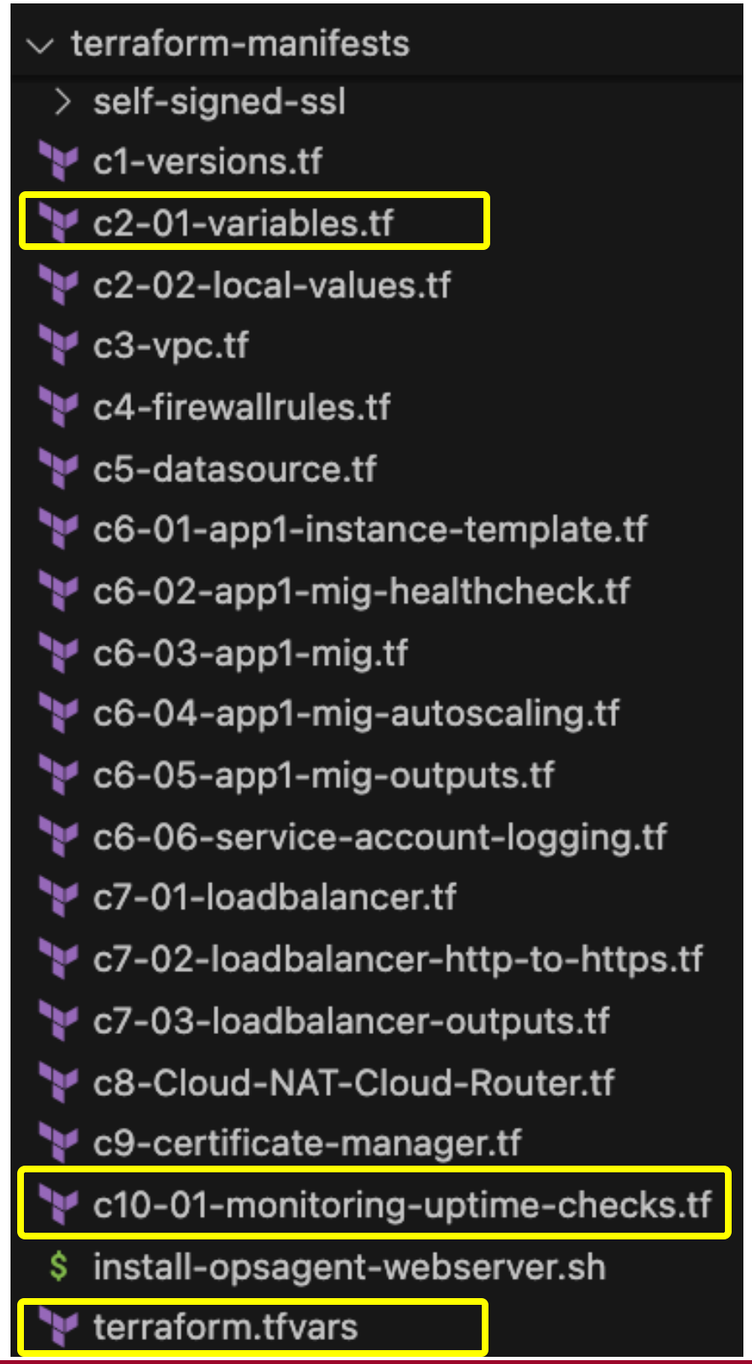
Application Monitoring

Cloud Monitoring

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTPS (Self-signed SSL) + Logging + Monitoring

Cloud Logging + Cloud Monitoring





What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Self-signed SSL) + Logging + Monitoring

NC	No Changes from C1, C3 to C9
C2-01	Create new input variable for GCP Notification channel
TF VARS	Update terraform.tfvars with GCP Notification channel input variable
C10-1	Create GCP Notification channel to send monitoring alerts
C10-1	Create Uptime check resource for myapp1 application with LB IP
C10-1	Create Monitoring alert policy if uptime check fails

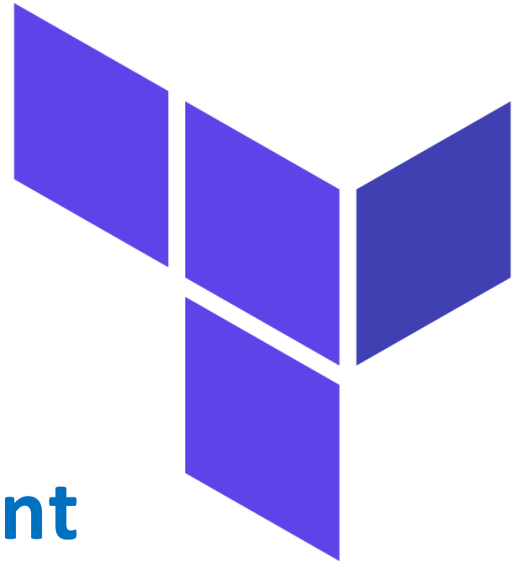
STACKSIMPLIFY

Demo-22

Terraform

GCP Cloud SQL

MySQL Database with Public Endpoint



Demo: Cloud SQL MySQL Database with Public Endpoint

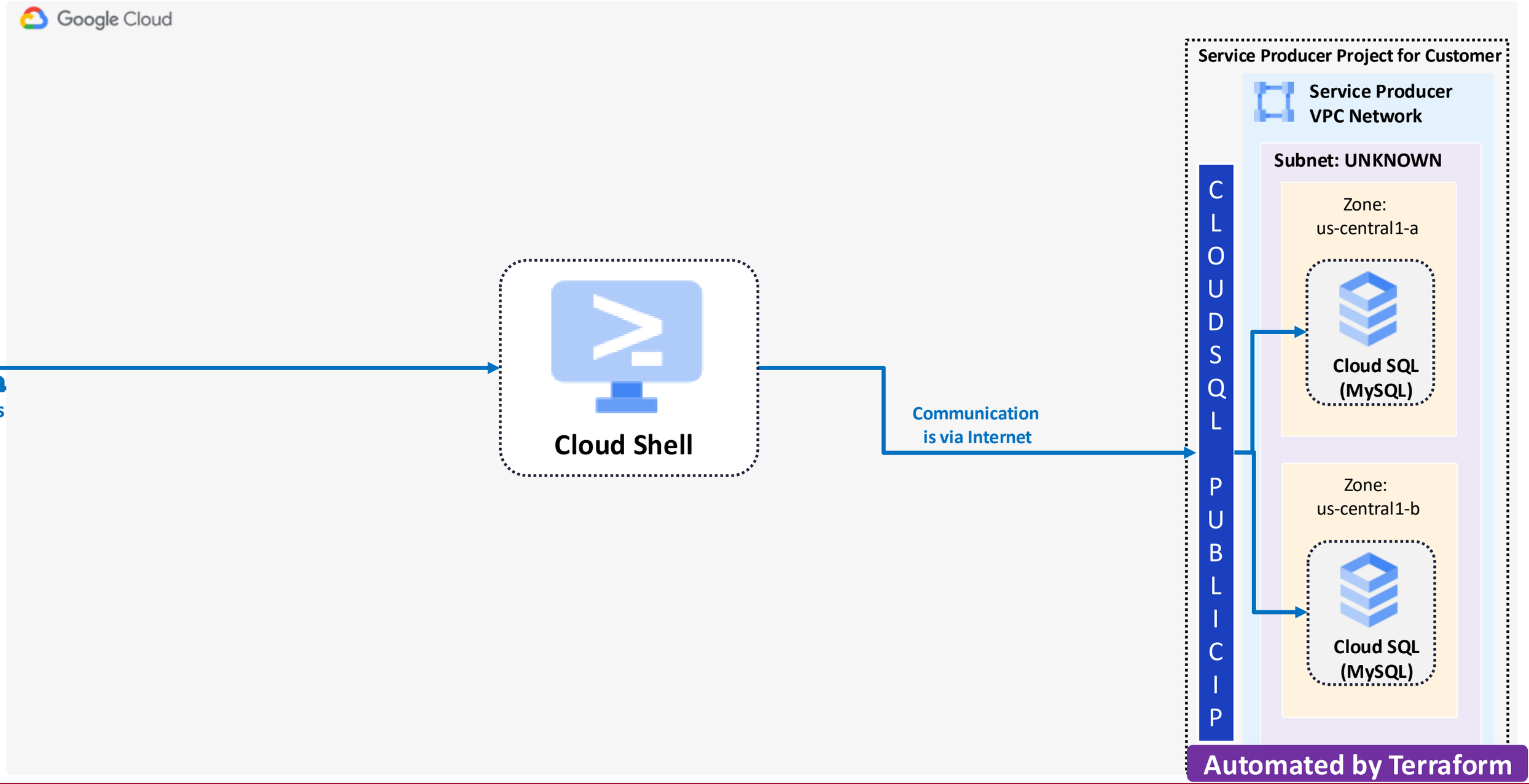
Terraform Remote Backends

- **Backend:** Place where terraform state files are stored
- Terraform uses persisted state data to keep track of the resources it manages
- **Backend Types**
  - **Local Backend:** Terraform working directory
  - **Remote Backend:** Stores state in remote location
    - AWS S3
    - Google Cloud Storage (GCS)
    - Azure Blob Storage
- **Google Cloud Storage Bucket (GCS)**
  - Bucket should be pre-created and ready to use
  - Recommended to enable Object versioning

```
# Terraform Settings Block
terraform {
  required_version = ">= 1.8"
  required_providers {
    google = {
      source = "hashicorp/google"
      version = ">= 5.26.0"
    }
  }
  backend "gcs" {
    bucket = "gcplearn9-tfstate"
    prefix = "cloudsql/publicdb"
  }
}
```

GCP Cloud SQL Public IP

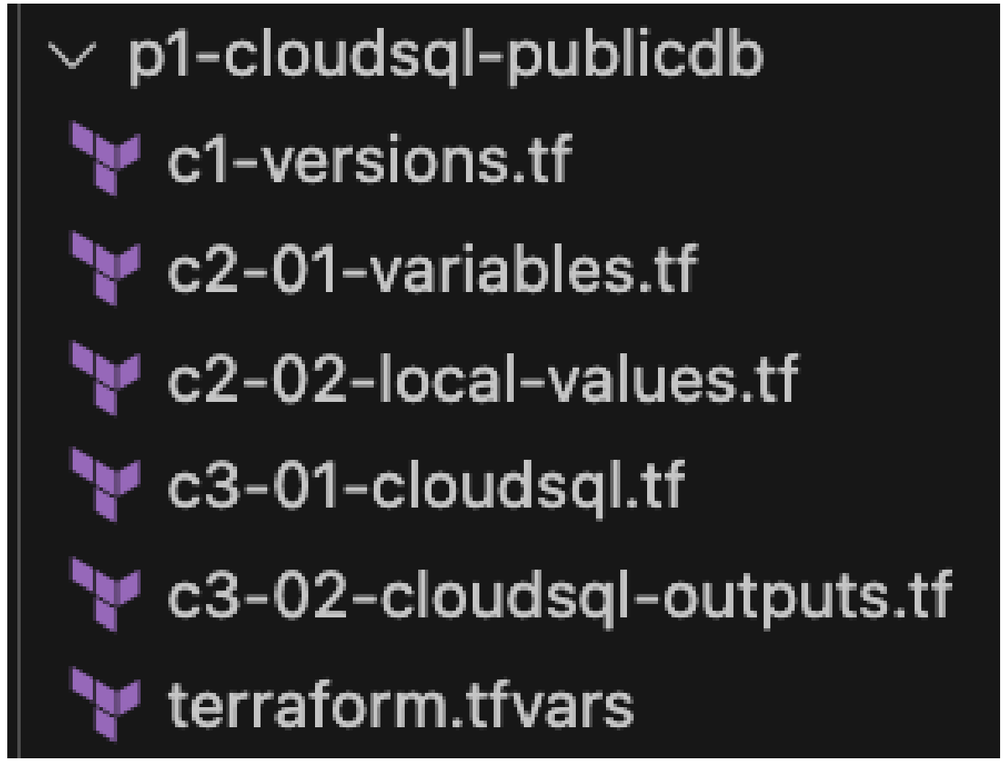
STACKSIMPLIFY



What are we going to learn?

STACKSIMPLIFY

Demo: Cloud SQL MySQL Database with Public Endpoint



C1	Define Remote backend as Cloud Storage Bucket
C2-01	Define Input Variables for Cloud SQL MySQL Database
C2-02	Define Local values
C3-01	Define Cloud SQL Database, Schema and User Terraform resources
C3-02	Define Cloud SQL Outputs

STACKSIMPLIFY

Terraform Remote State Data source

Terraform Template Functions

Demo-23

Terraform

DNS to DB

Cloud SQL Public IP

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTPS **(Self-signed SSL)** + Logging + Monitoring + **Cloud SQL (Public IP)**

Terraform Remote State Datasource

STACKSIMPLIFY

The terraform_remote_state data source retrieves the root module output values from project-1 Terraform configuration, using the latest state snapshot from the remote backend.

Project-1

Terraform Resources
1. Cloud SQL Database

cloudsql/publicdb/default.tfstate

terraform_remote_state
data source

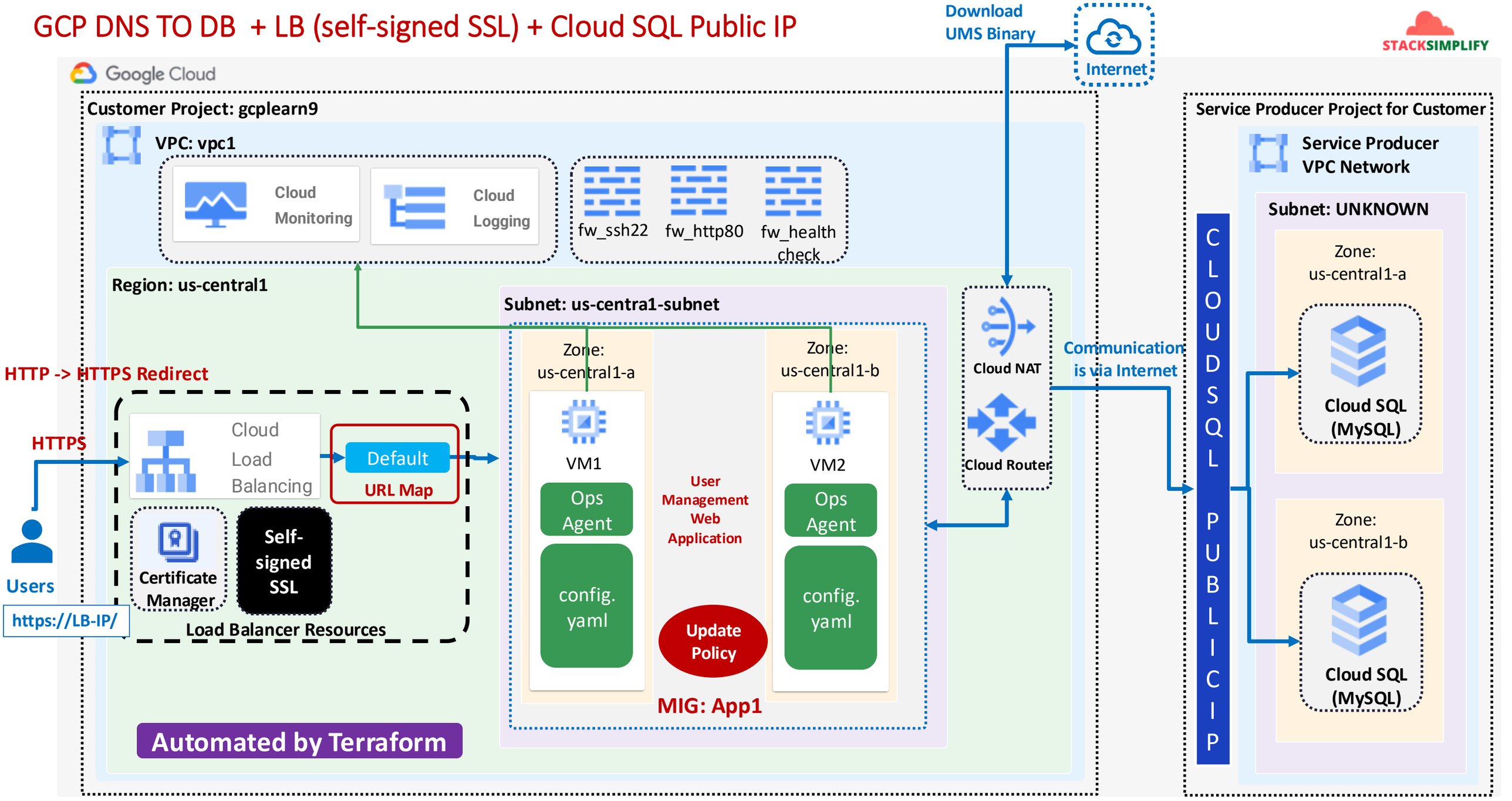
Outputs from Project-1
1. Cloud SQL Public IP

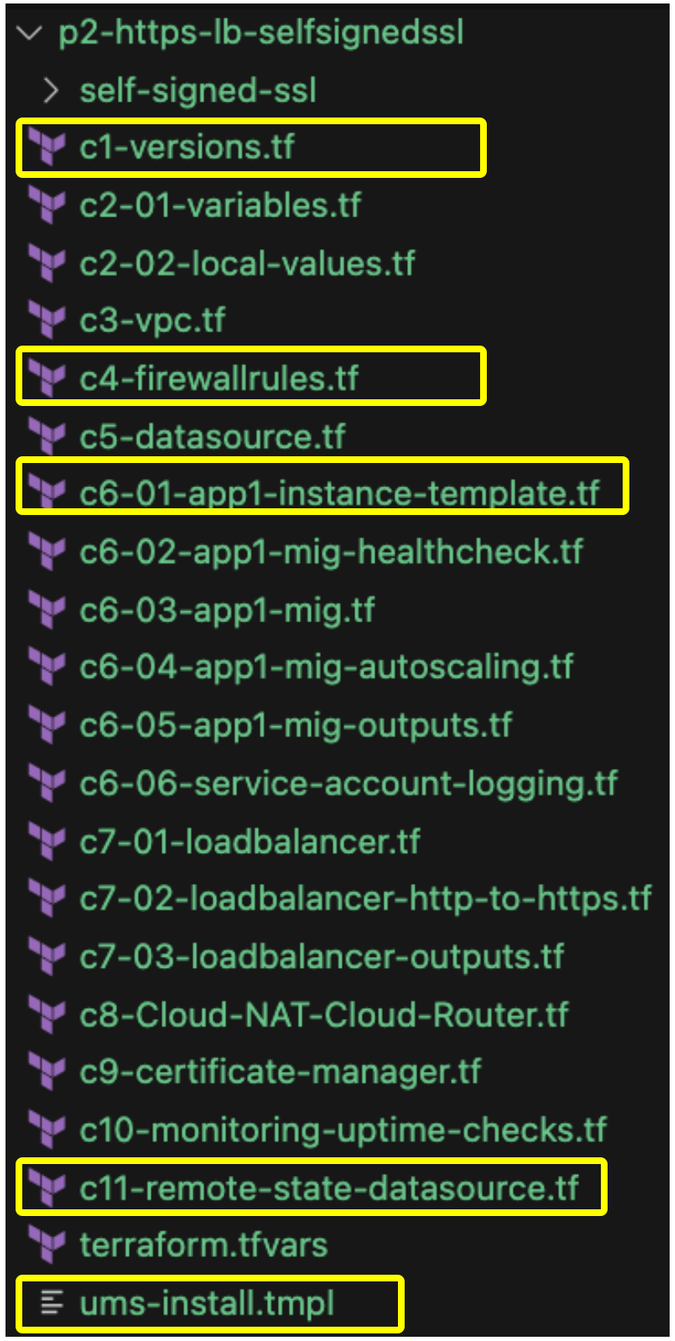
Project-2

Terraform Resources
1. **VPC, Subnets**
2. **Instance Templates, MIG**
3. **Service Account**
4. **Load Balancer**
5. **Cloud NAT, Cloud Router**
6. **Certificate Manager**
7. **Uptime Checks**

myapp1/httpslb-selfsigned-publicdb/default.tfstate

GCP DNS TO DB + LB (self-signed SSL) + Cloud SQL Public IP



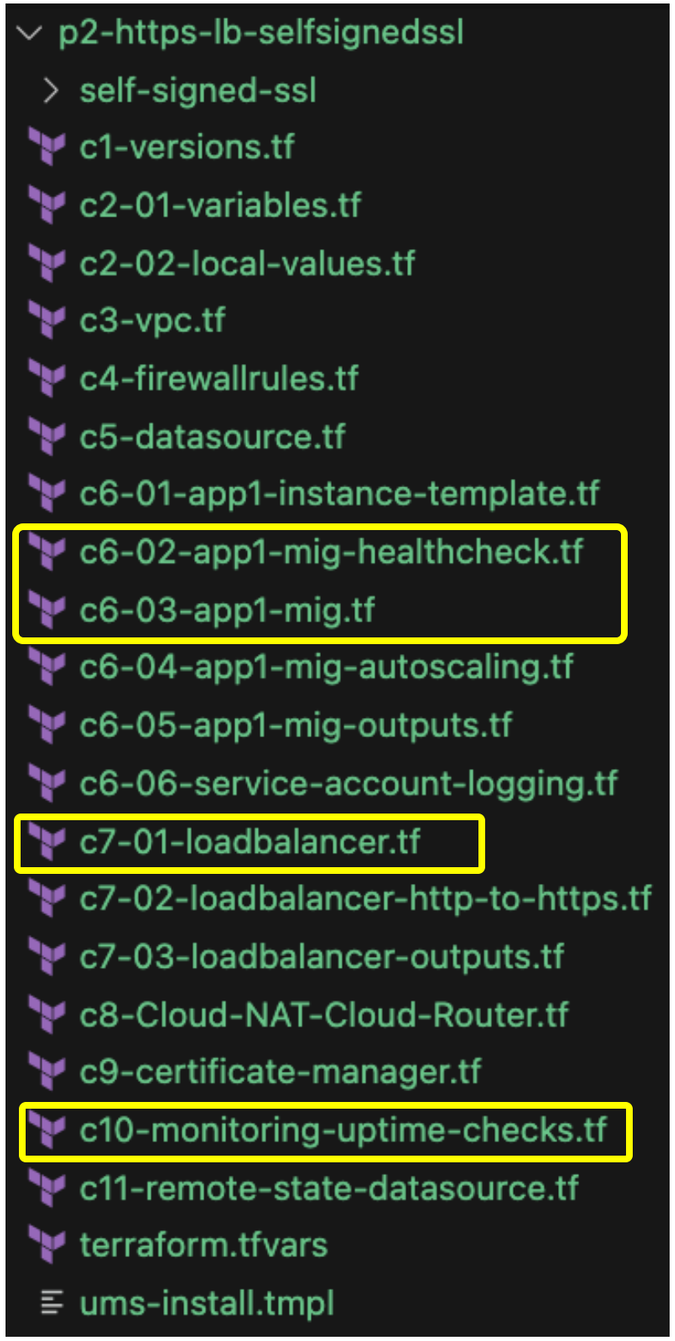


What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS **(Self-signed SSL)** + Logging + Monitoring + **Cloud SQL (Public IP)**

NC	No Changes from C2 to C3, C6-04 to C6-06, C7-02 to C9
C1	Define remote backend as Cloud Storage Bucket
C11	Define Remote State Data source to access Cloud SQL Database from Project-1
C4	Define firewall rules for UMS Application (Port 8080)
UMS Install	Review ums-install.tmpl to install UMS Application using startup-script + Terraform Template Functions
C6-01	Update instance template with startup-script and implement Terraform Template functions



What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Self-signed SSL) + Logging + Monitoring + Cloud SQL (Public IP)

C6-02	Update health check with UMS Port 8080 and Path as /login
C6-03	Update named port to appserver with port as 8080
C7-01	Update Load balancer health check with port 8080 and Path as /login
C7-01	Update backend service with session affinity, and port name as appserver
C10	Update uptime checks with path as “/login” and content as “Username”

STACKSIMPLIFY

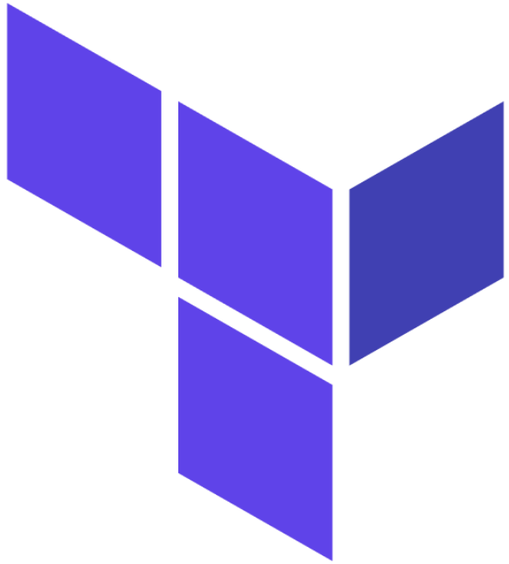
Pre-requisite:
Registered Domain is required to implement this demo

Demo-24

Terraform

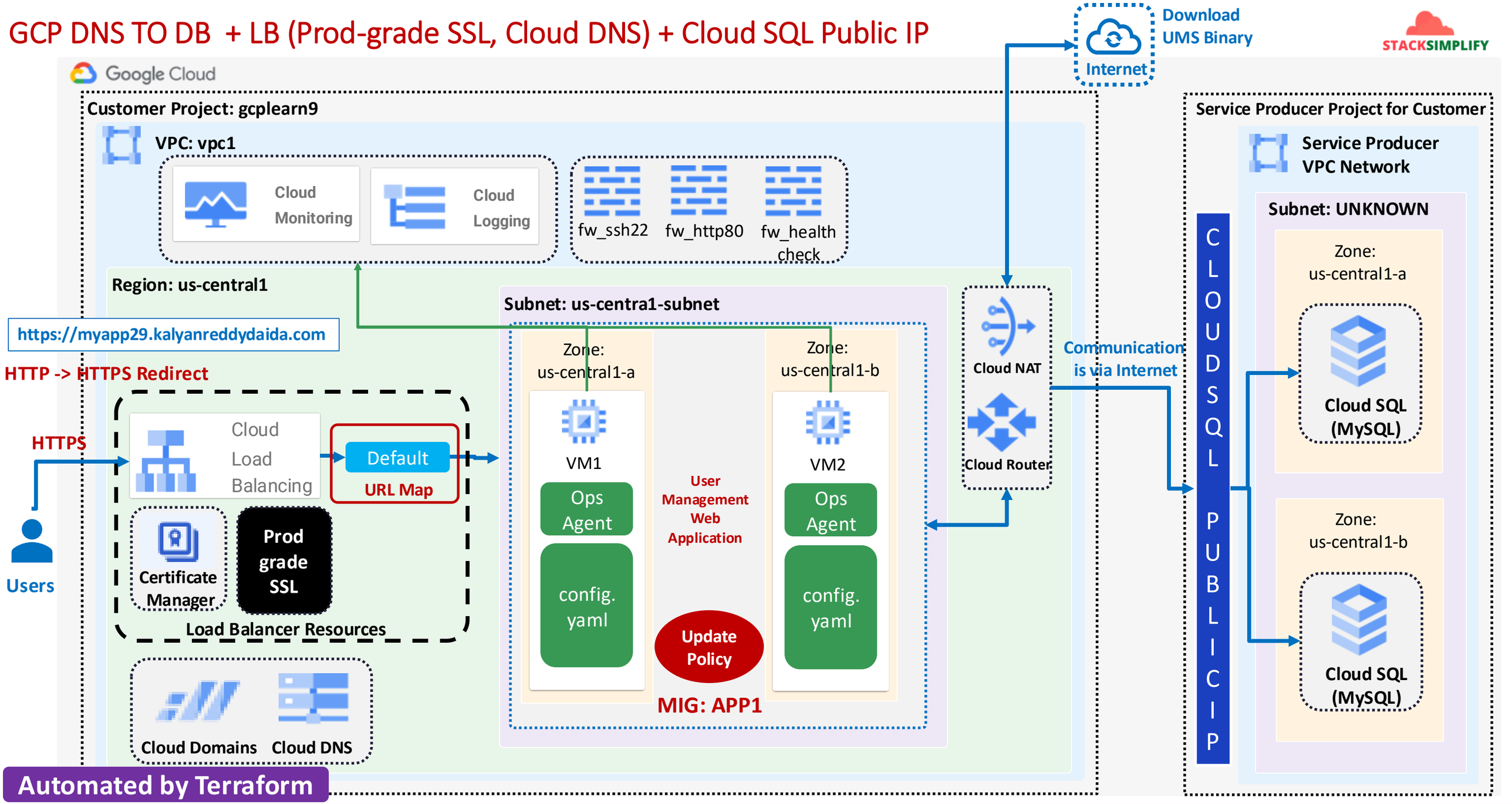
DNS to DB

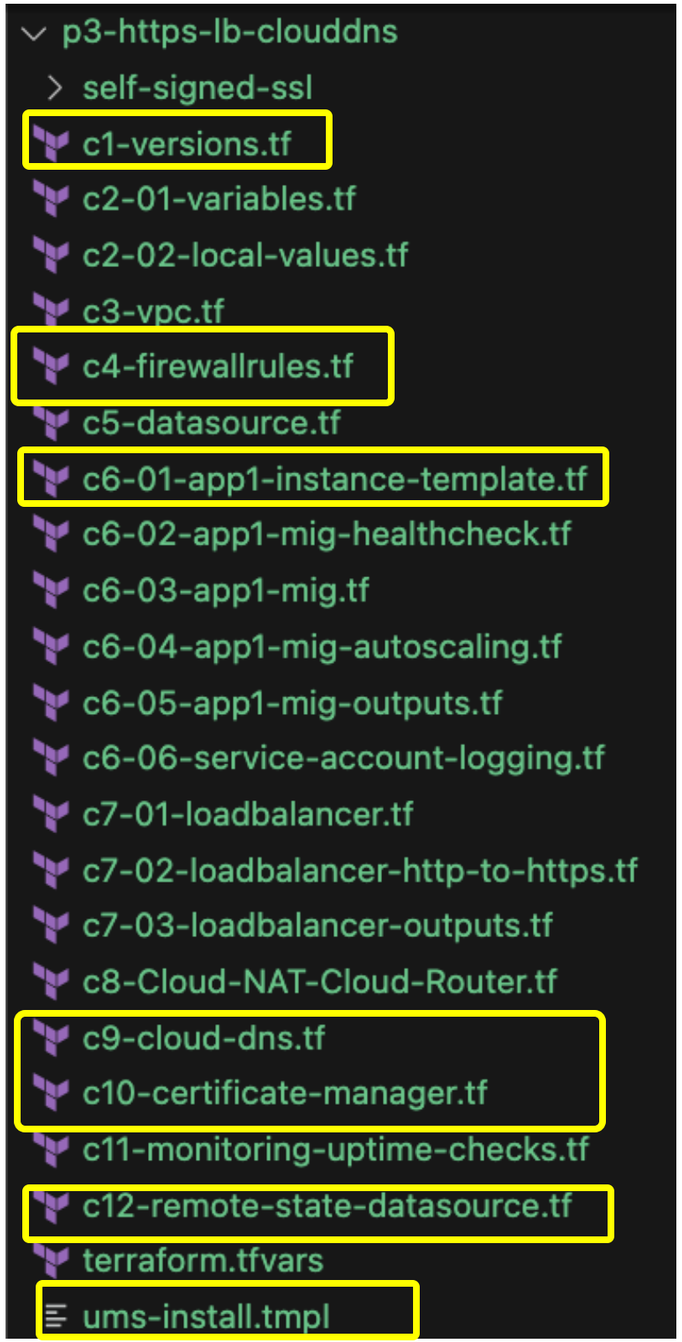
Cloud SQL Public IP + Cloud Domains + Cloud DNS



Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Prod grade SSL, Cloud DNS) + Logging + Monitoring + Cloud SQL (Public IP)

GCP DNS TO DB + LB (Prod-grade SSL, Cloud DNS) + Cloud SQL Public IP



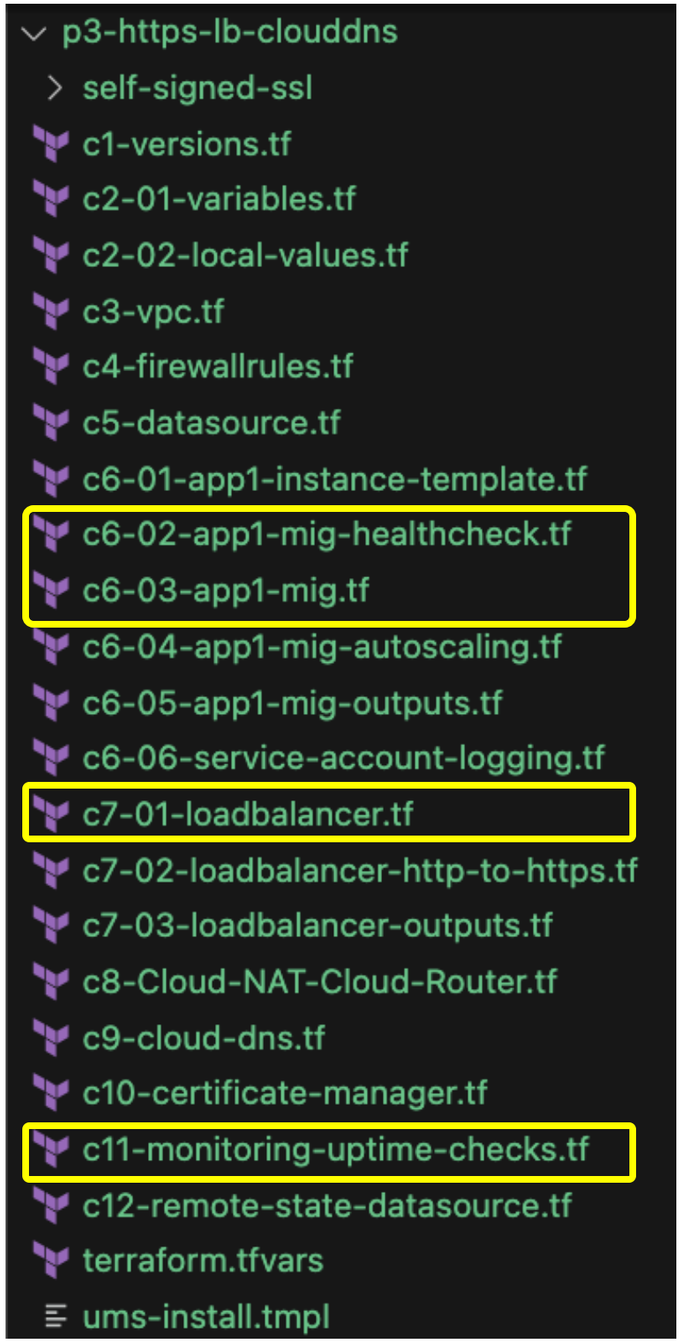


What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Prod grade SSL, Cloud DNS) + Logging + Monitoring + Cloud SQL (Public IP)

NC	No Changes from C2 to C3, C6-04 to C6-06, C7-02 to C10
C1	Define remote backend as Cloud Storage Bucket
C12	Define Remote State Data source to access Cloud SQL Database from Project-1
C4	Define firewall rules for UMS Application (Port 8080)
UMS Install	Review ums-install.tmpl to install UMS Application using startup-script + Terraform Template Functions
C6-01	Update instance template with startup-script and implement Terraform Template functions
C9, C10	Files related to Cloud DNS and Prod grade ssl certificates with DNS Auth from Demo-16



What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Prod grade SSL, Cloud DNS) + Logging + Monitoring + Cloud SQL (Public IP)

C6-02	Update health check with UMS Port 8080 and Path as /login
C6-03	Update named port to appserver with port as 8080
C7-01	Update Load balancer health check with port 8080 and Path as /login
C7-01	Update backend service with session affinity, and port name as appserver
C11	Update uptime checks with path as “/login” and content as “Username”

Demo-24

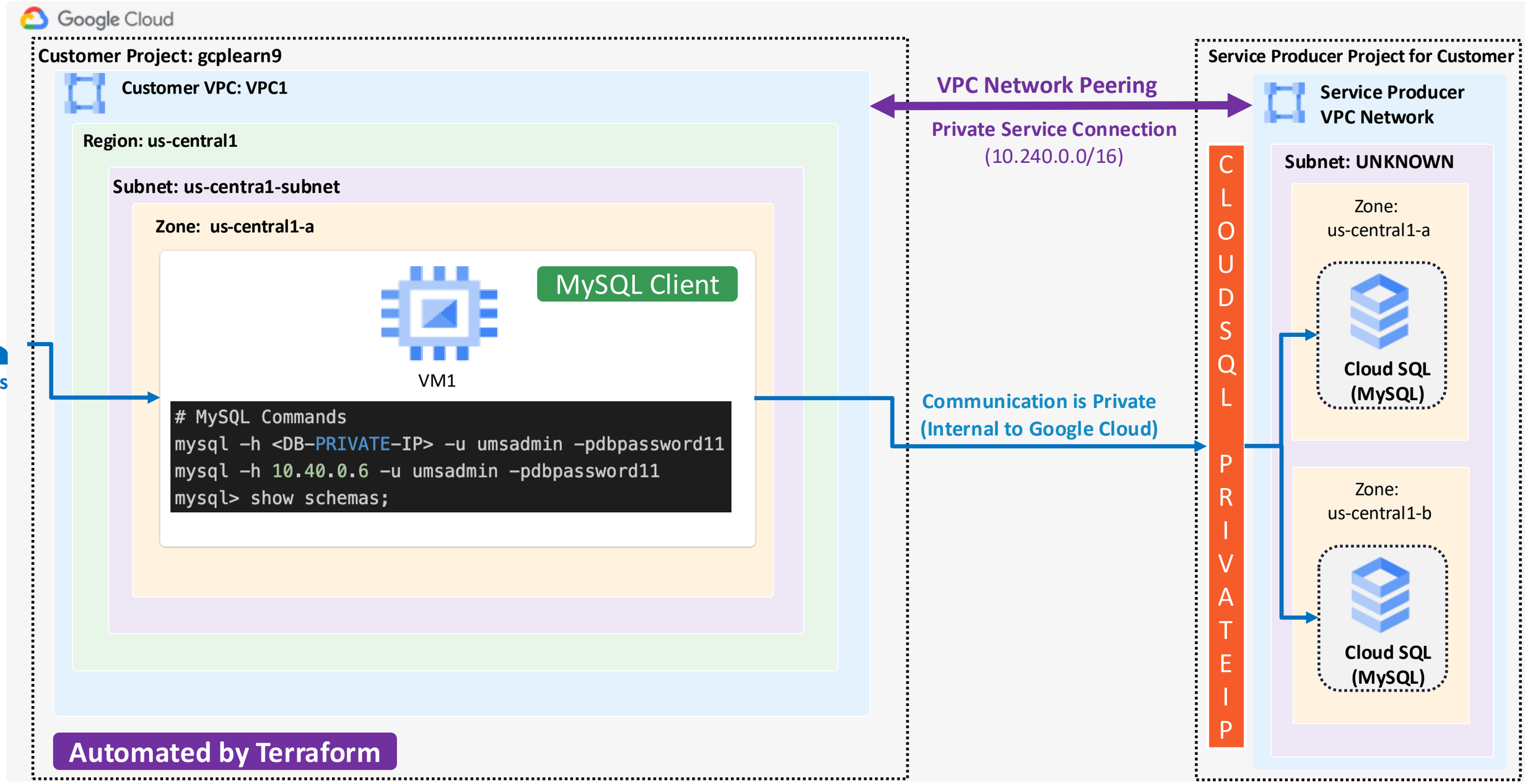
Terraform

GCP Cloud SQL

Cloud SQL with Private Service Connection

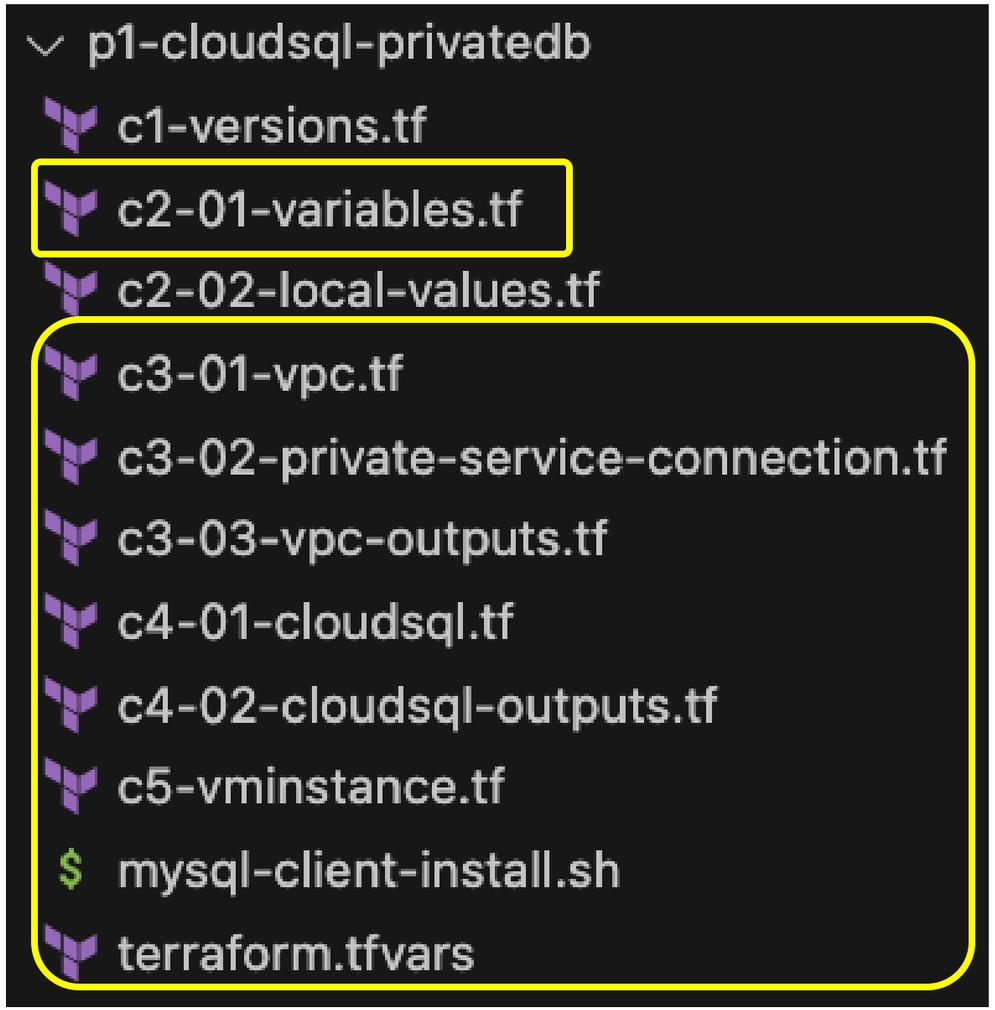
Demo: Cloud SQL MySQL with Private Service Connection

GCP Cloud SQL Private IP



What are we going to learn?

STACKSIMPLIFY



Demo: Cloud SQL MySQL with Private Service Connection

C2	Added Variable related to MySQL DB version, updated in terraform.tfvars
C3-01	VPC, Subnet and Regional Proxy Subnet for Regional Load Balancer (Same as previous demos)
C3-02	**NEW:** Two resources(1. Reserve Private IP Range for PSC, 2. Private Service Connection Resource)
C3-03	No changes in VPC Outputs from previous demos
C4-01	Create Cloud SQL Database with private network enabled.
C4-02	Define private IP in Cloud SQL Outputs
C5	Create a VM Instance in VPC with mysql client installed to test private connection to Cloud SQL DB

STACKSIMPLIFY

Demo-26

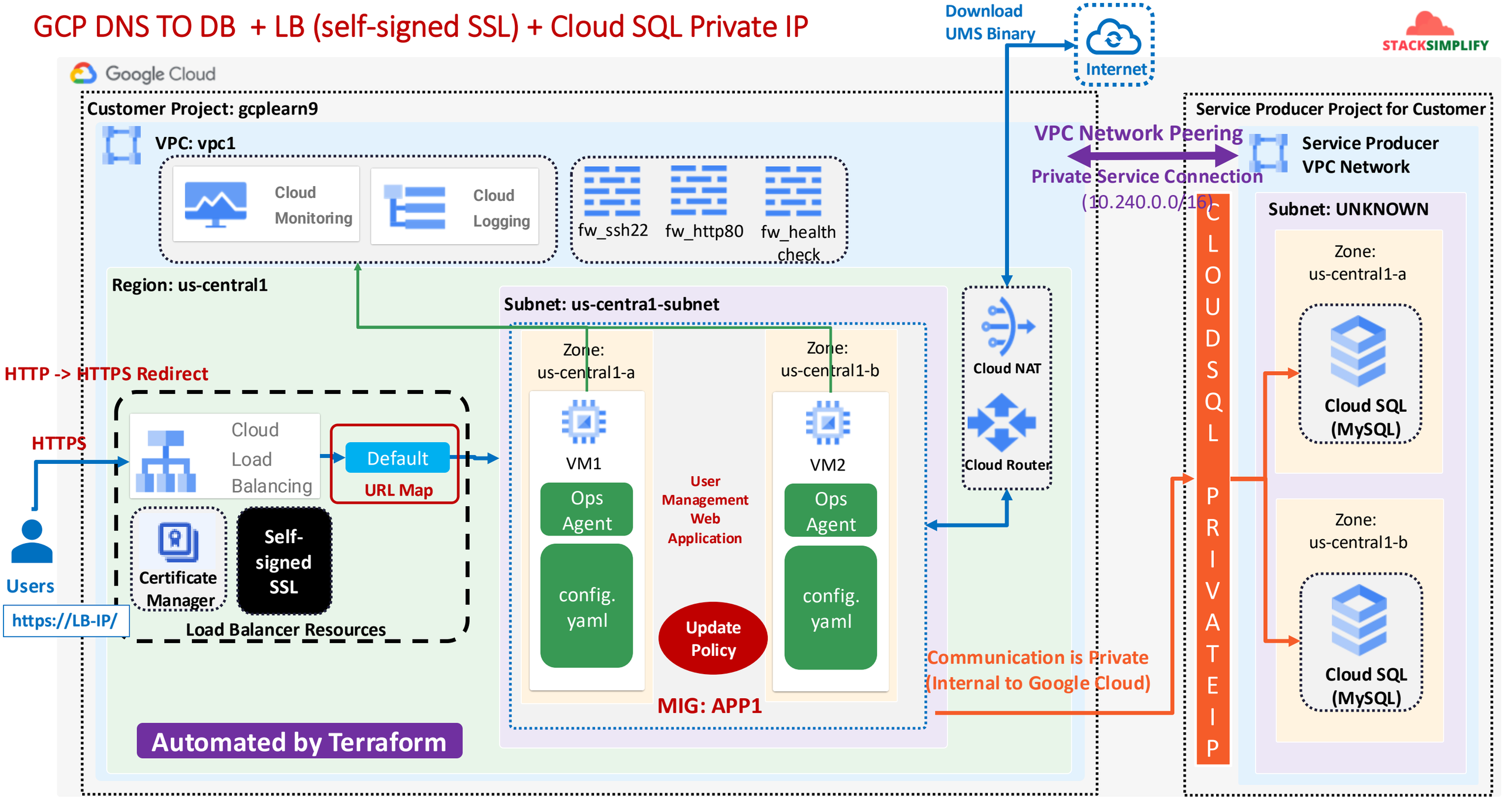
Terraform

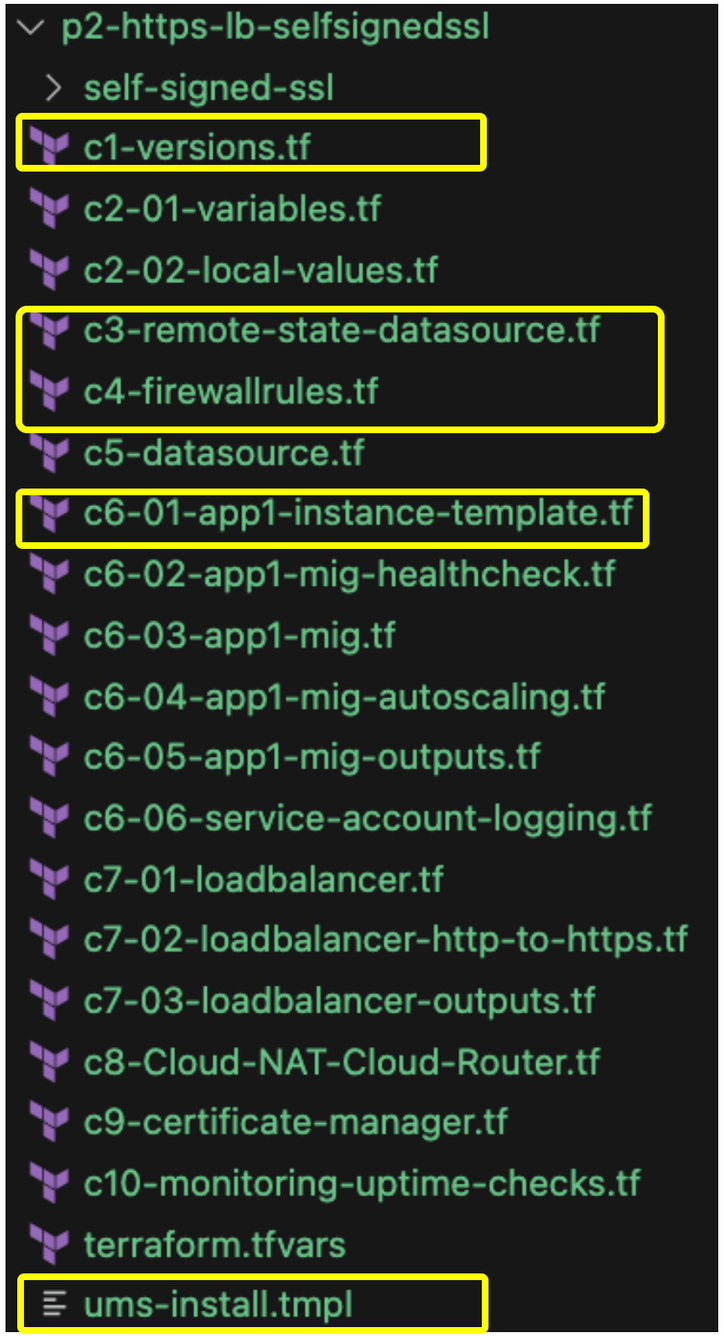
DNS to DB

Cloud SQL with Private Service Connection

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTPS **(Self-signed SSL)** + Logging + Monitoring + **Cloud SQL (Private IP)**

GCP DNS TO DB + LB (self-signed SSL) + Cloud SQL Private IP



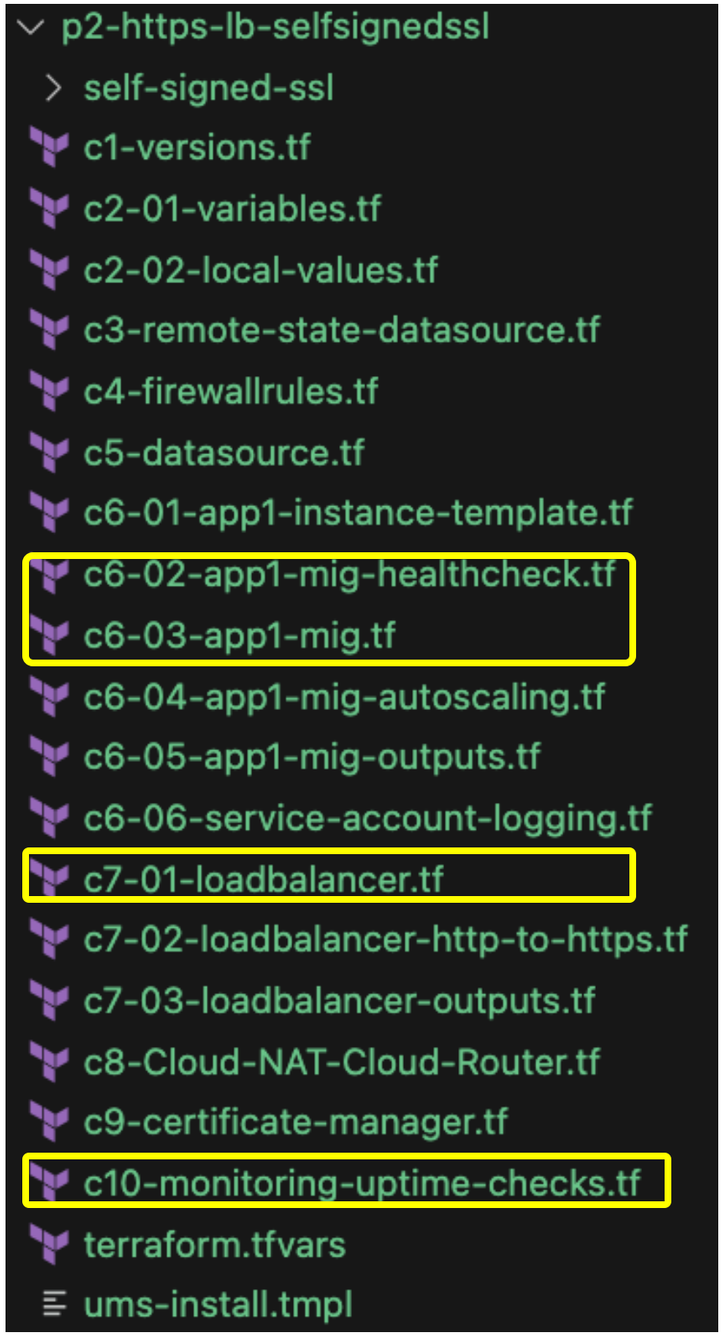


What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS **(Self-signed SSL)** + Logging + Monitoring + **Cloud SQL (Private IP)**

NC	No Changes from C2 to C3, C6-04 to C6-06, C7-02 to C9
C1	Define remote backend as Cloud Storage Bucket
C3	Define Remote State Data source to access Cloud SQL Database from Project-1
C4	Define firewall rules for UMS Application (Port 8080)
UMS Install	Review ums-install.tmpl to install UMS Application using startup-script + Terraform Template Functions
C6-01	Update instance template with startup-script and implement Terraform Template functions



What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Self-signed SSL) + Logging + Monitoring + Cloud SQL (Private IP)

C6-02	Update health check with UMS Port 8080 and Path as /login
C6-03	Update named port to appserver with port as 8080
C7-01	Update Load balancer health check with port 8080 and Path as /login
C7-01	Update backend service with session affinity, and port name as appserver
C10	Update uptime checks with path as “/login” and content as “Username”

STACKSIMPLIFY

Pre-requisite:
Registered Domain is required to implement this demo

Demo-27

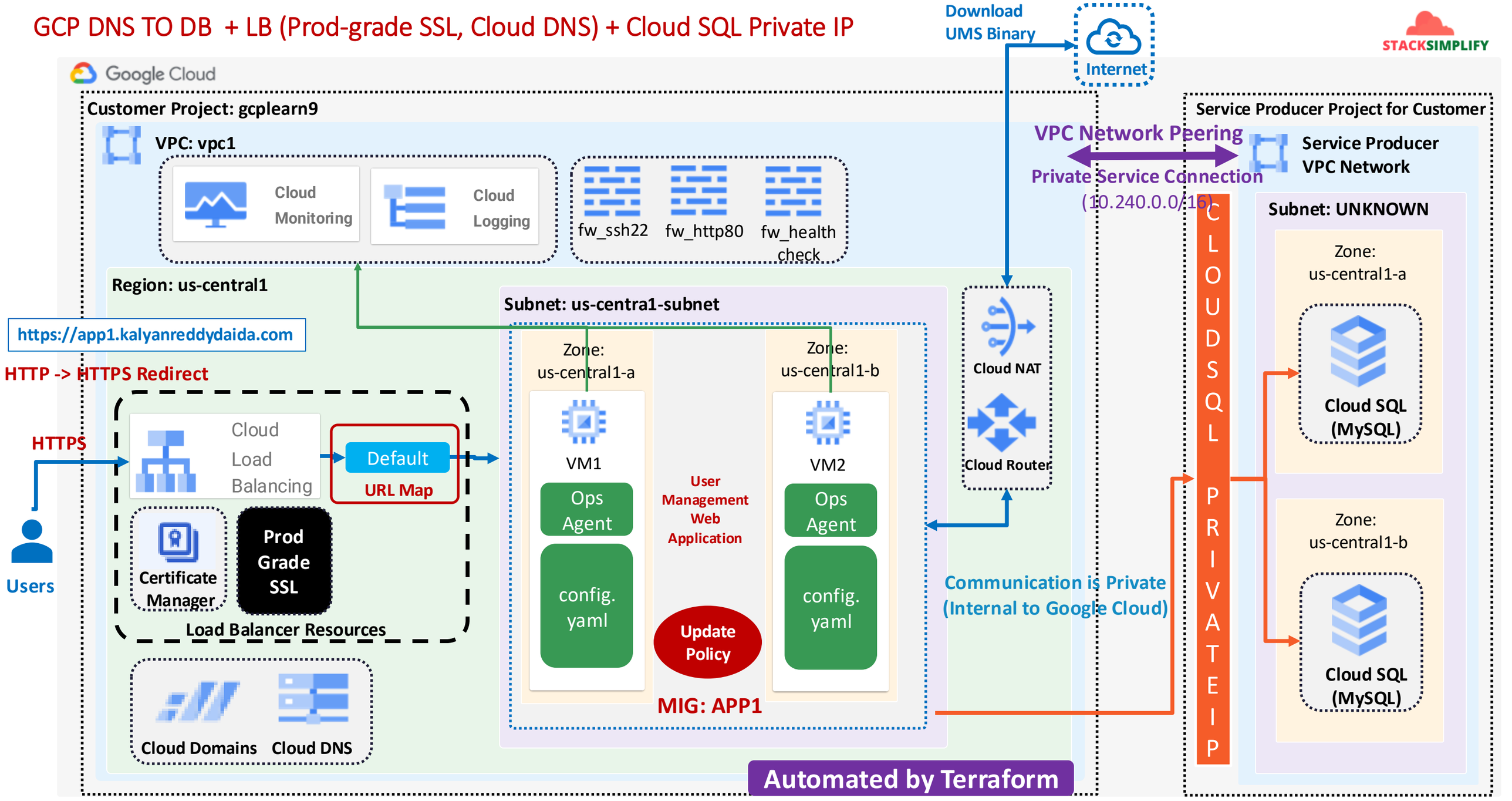
Terraform

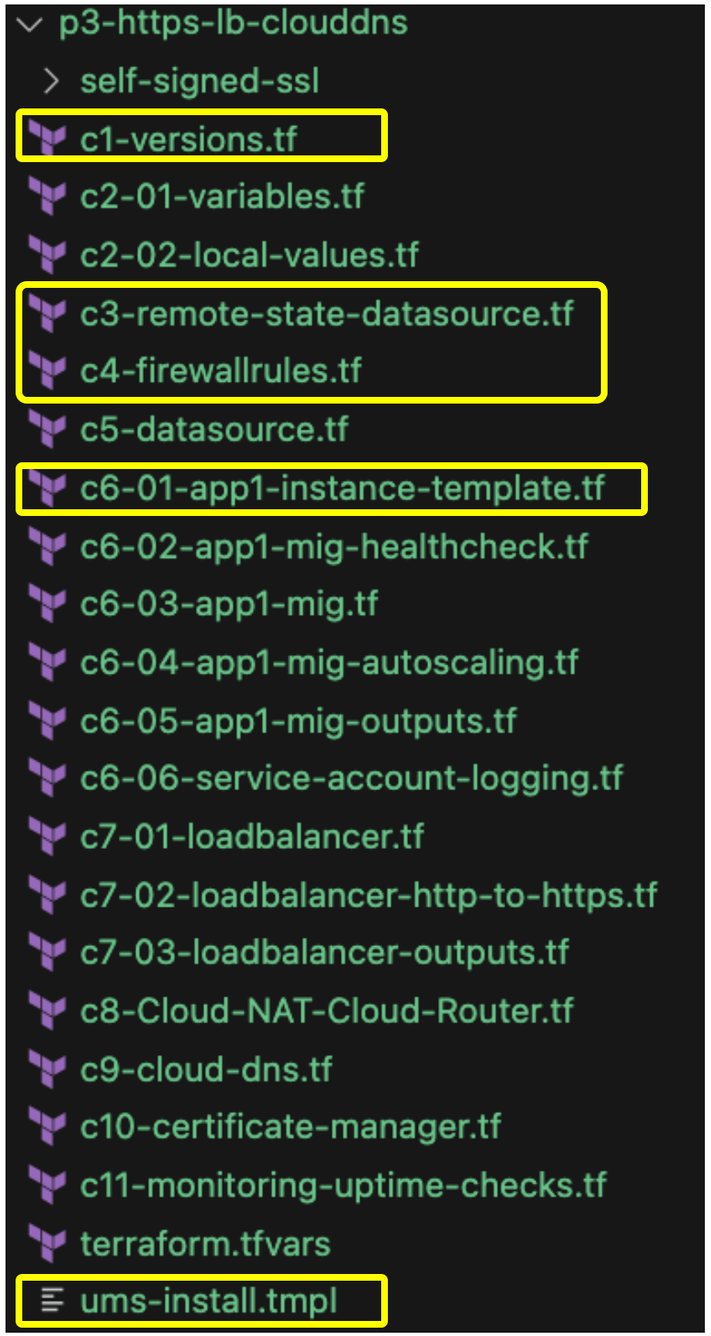
DNS to DB

Cloud SQL with Private Service Connection

Demo: Custom VPC + Regional MIG Private IP +
Regional Application Load Balancer HTTPS **(Prod grade SSL, Cloud DNS)** + Logging + Monitoring +
Cloud SQL (Private IP)

GCP DNS TO DB + LB (Prod-grade SSL, Cloud DNS) + Cloud SQL Private IP



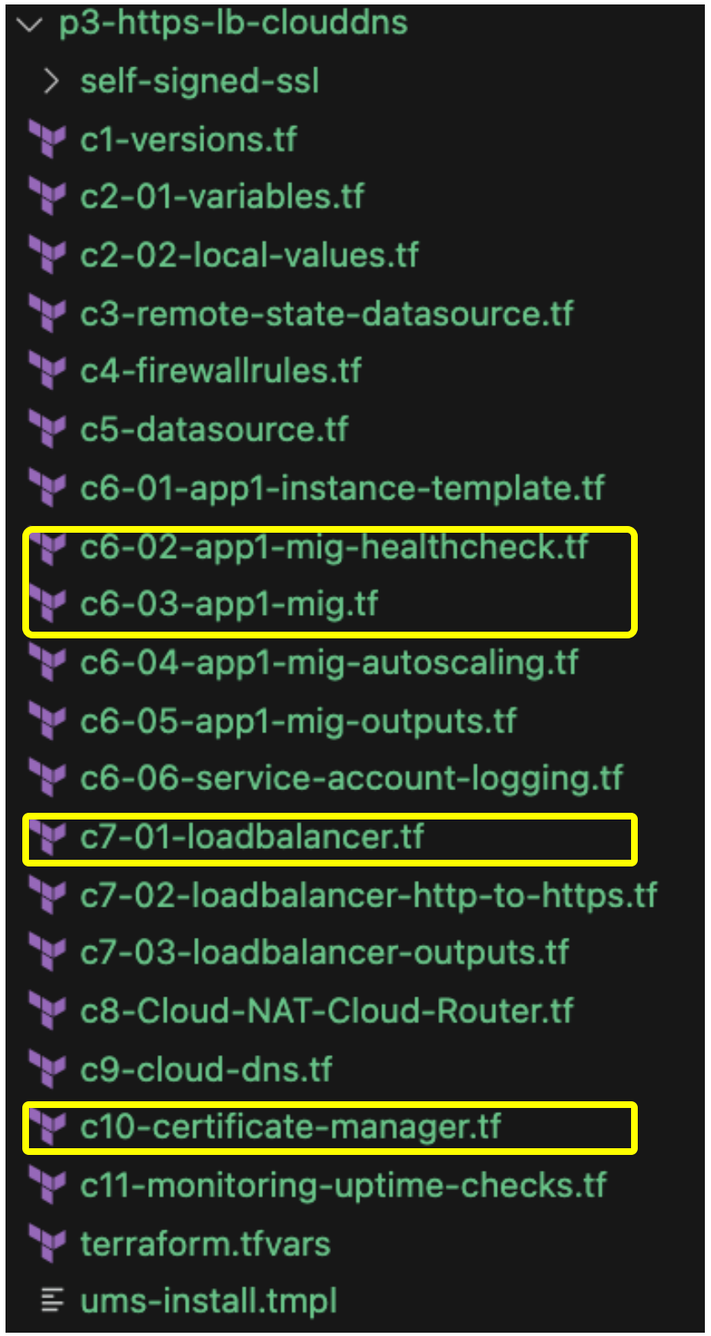


What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Prod grade SSL, Cloud DNS) + Logging + Monitoring + Cloud SQL (Private IP)

NC	No Changes from C2 to C3, C6-04 to C6-06, C7-02 to C9
C1	Define remote backend as Cloud Storage Bucket
C3	Define Remote State Data source to access Cloud SQL Database from Project-1
C4	Define firewall rules for UMS Application (Port 8080)
UMS Install	Review ums-install.tmpl to install UMS Application using startup-script + Terraform Template Functions
C6-01	Update instance template with startup-script and implement Terraform Template functions



What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Regional MIG Private IP + Regional Application Load Balancer HTTPS (Prod grade SSL, Cloud DNS) + Logging + Monitoring + Cloud SQL (Private IP)

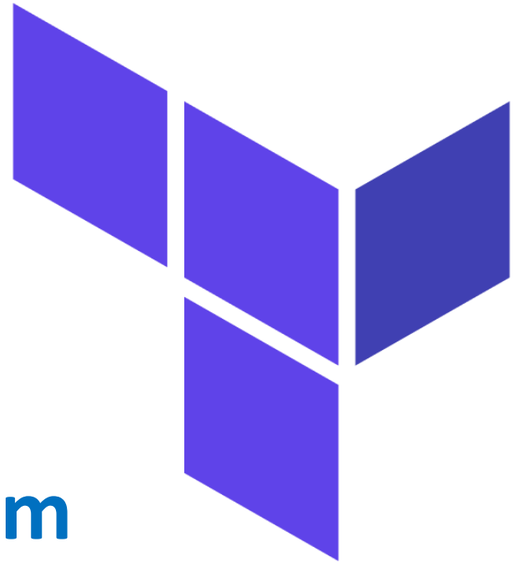
C6-02	Update health check with UMS Port 8080 and Path as /login
C6-03	Update named port to appserver with port as 8080
C7-01	Update Load balancer health check with port 8080 and Path as /login
C7-01	Update backend service with session affinity, and port name as appserver
C10	Update uptime checks with path as “/login” and content as “Username”

STACKSIMPLIFY

Demo-28

Terraform Modules

Use pre-built Modules from Terraform Registry

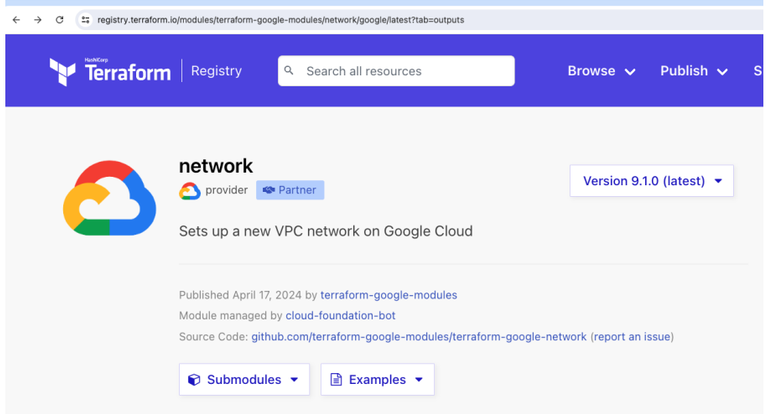


Demo: Custom VPC **(Uses Terraform Module from Terraform Public Registry)** + Firewall Rules + VM Instance

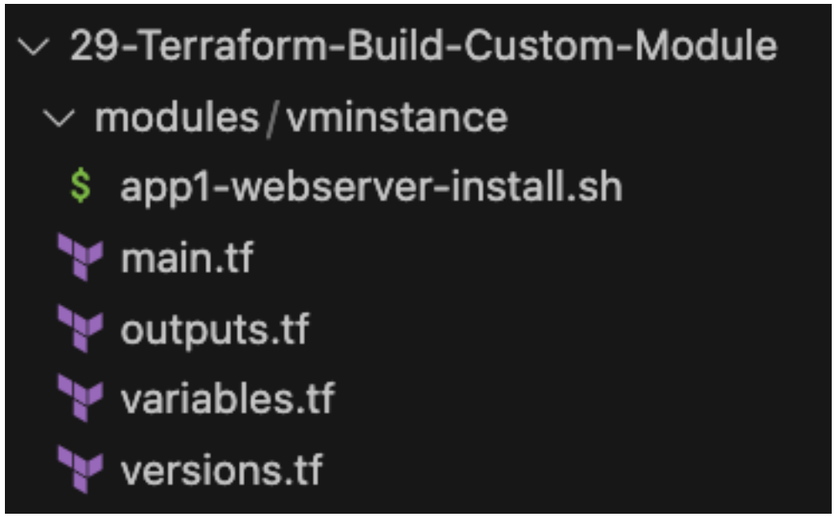
Terraform Modules

- A module consists of a collection of .tf files kept together in a directory.
- Modules are the main way to package and reuse resource configurations with Terraform.
- We can compare modules to functions in our standard programming languages (define once and reuse wherever needed)
- **Example:**
  - Define a VPC Module once, publish it to a Terraform registry
  - Call the VPC module wherever needed (reuse it)

Public Module on Terraform Registry

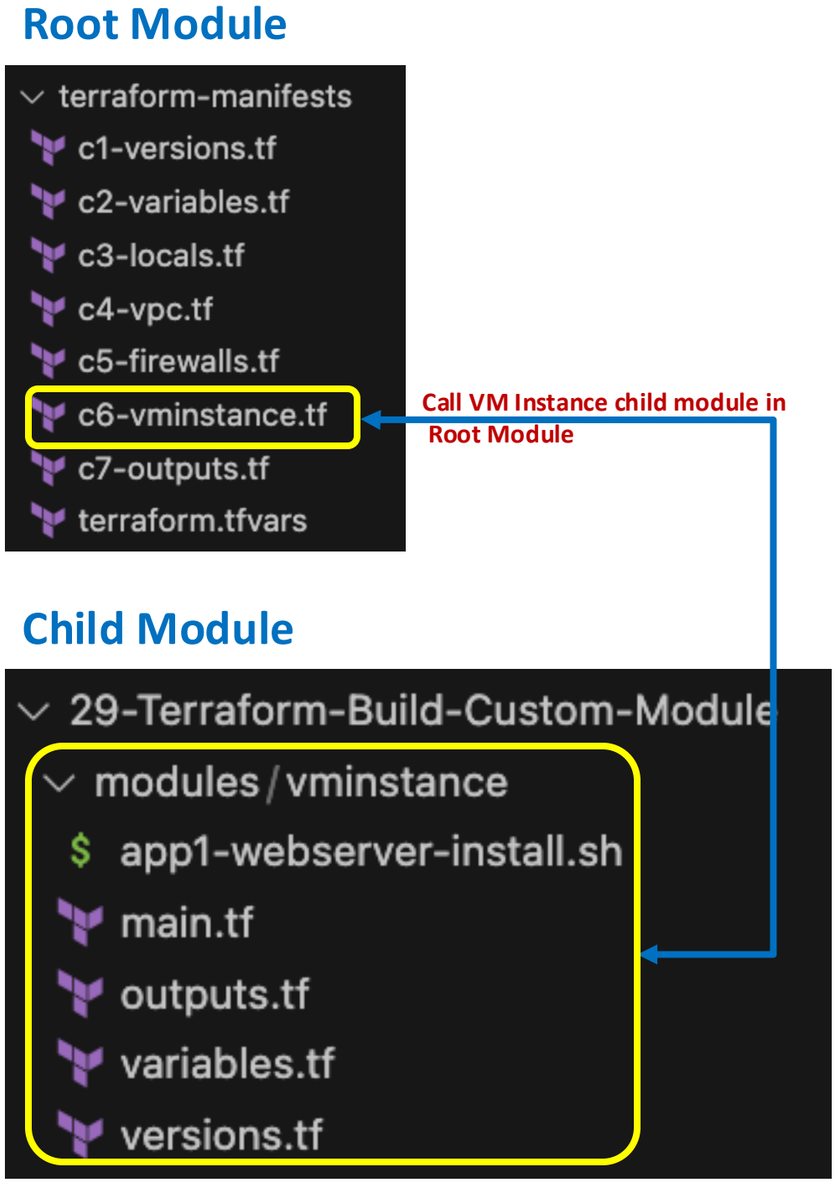


Custom Module



Terraform Modules

- **Root Module:** Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the .tf files in the main working directory
- A Terraform module (usually the root module of a configuration) can call other modules to include their resources into the configuration.
- **Child Module:** A module that has been called by another module is often referred to as a child module.
- Child modules can be called multiple times within the same configuration, and multiple configurations can use the same child module.



Terraform Modules

STACKSIMPLIFY

- **Module Types**
  - **Local Modules (Local Paths)**
    - Accessible using folder path locally
    - Accessible locally in that desktop or for others via shared drives
  - **Public Modules**
    - Modules published to Terraform Registry
    - Publicly accessible to anyone (Open source)
    - Free to use, use with caution, will contain continuous upgrades and might break our current code
    - If used, need to follow strong version constraints or fixed module versions to avoid major outages during upgrades
  - **Private Modules**
    - Modules developed and published to private registry in their respective organizations
    - Terraform Registry also supports private registry concept
    - Accessible only to that respective organization users (Highly secure, internal to that organization)

Terraform Modules

- **Module Storage and Access:** The module installer supports installation from several different source types.
  - Local Paths
  - Terraform Registry
  - GitHub
  - HTTP URLs
  - S3 Buckets
  - GCS Buckets
  - and many more

Local paths

```
module "consul" {
  source = "./consul"
}
```

TF Registry

```
module "vpc" {
    source  = "terraform-google-modules/network/google"
    version = "~> 9.1"
```

GitHub

```
module "consul" {
  source = "github.com/hashicorp/example"
}
```

HTTP URL

```
module "vpc" {
  source = "https://example.com/vpc-module.zip"
}
```

S3 Bucket

```
module "consul" {
  source = "s3::https://s3-eu-west-1.amazonaws.com/examplecorp-terraform-modules/vpc.
}
```

GCS Bucket

```
module "consul" {
  source = "gcs::https://www.googleapis.com/storage/v1/modules/foomodule.zip"
}
```

Terraform Registry - Use Publicly Available Modules

The Terraform Registry hosts a broad collection of publicly available Terraform modules for configuring many kinds of common infrastructure.

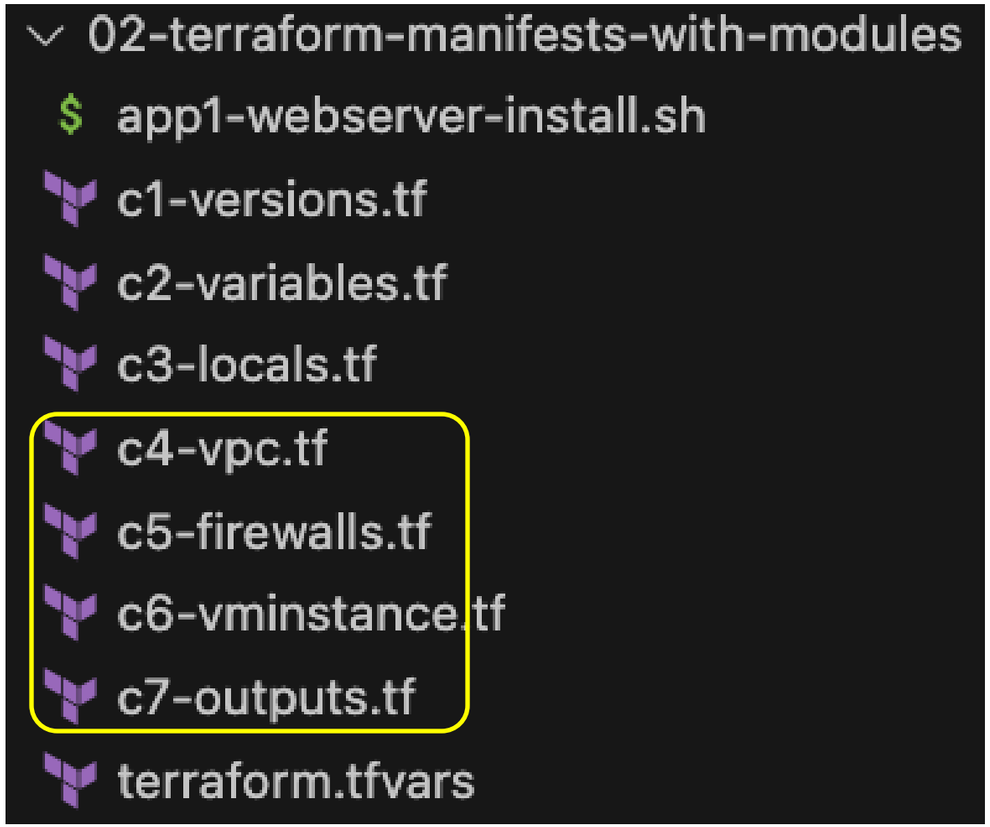
Modules Demo 1

These modules are free to use, and Terraform can download them automatically if you specify the appropriate source and version in a module call block.

```
# Module: VPC
module "vpc" {
    source  = "terraform-google-modules/network/google"
    version = "~> 9.1"
    project_id   = var.gcp_project
    network_name = "${local.name}-vpc"
    routing_mode = "GLOBAL"
    subnets = [
        {
            subnet_name           = "${local.name}-${var.gcp_region1}-subnet"
            subnet_ip             = "10.128.0.0/20"
            subnet_region         = var.gcp_region1
        }
    ]
}
```

What are we going to learn?

STACKSIMPLIFY



Demo: Custom VPC (Uses Terraform Module from Terraform Public Registry) + Firewall Rules + VM Instance

NC	No Changes from C1, C2, C3
C4	Create VPC Module using Terraform public registry
C5	Update firewall rules with VPC ID from VPC Module
C6	Update Subnet with Subnet ID from VPC Module
C7	Update VPC ID and Subnet ID from VPC Module

STACKSIMPLIFY

Demo-29

Terraform Modules

Create Custom or Local Terraform Module for VM Instance Resource

Demo: Custom VPC + Firewall Rules + VM Instance **(Custom or Local Terraform Module)**

Build a Local Terraform Module

STACKSIMPLIFY

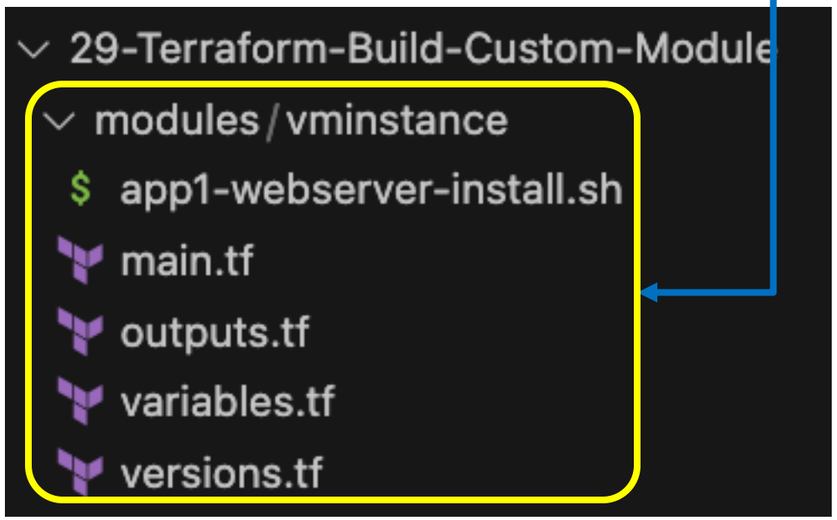
Modules
Demo 2

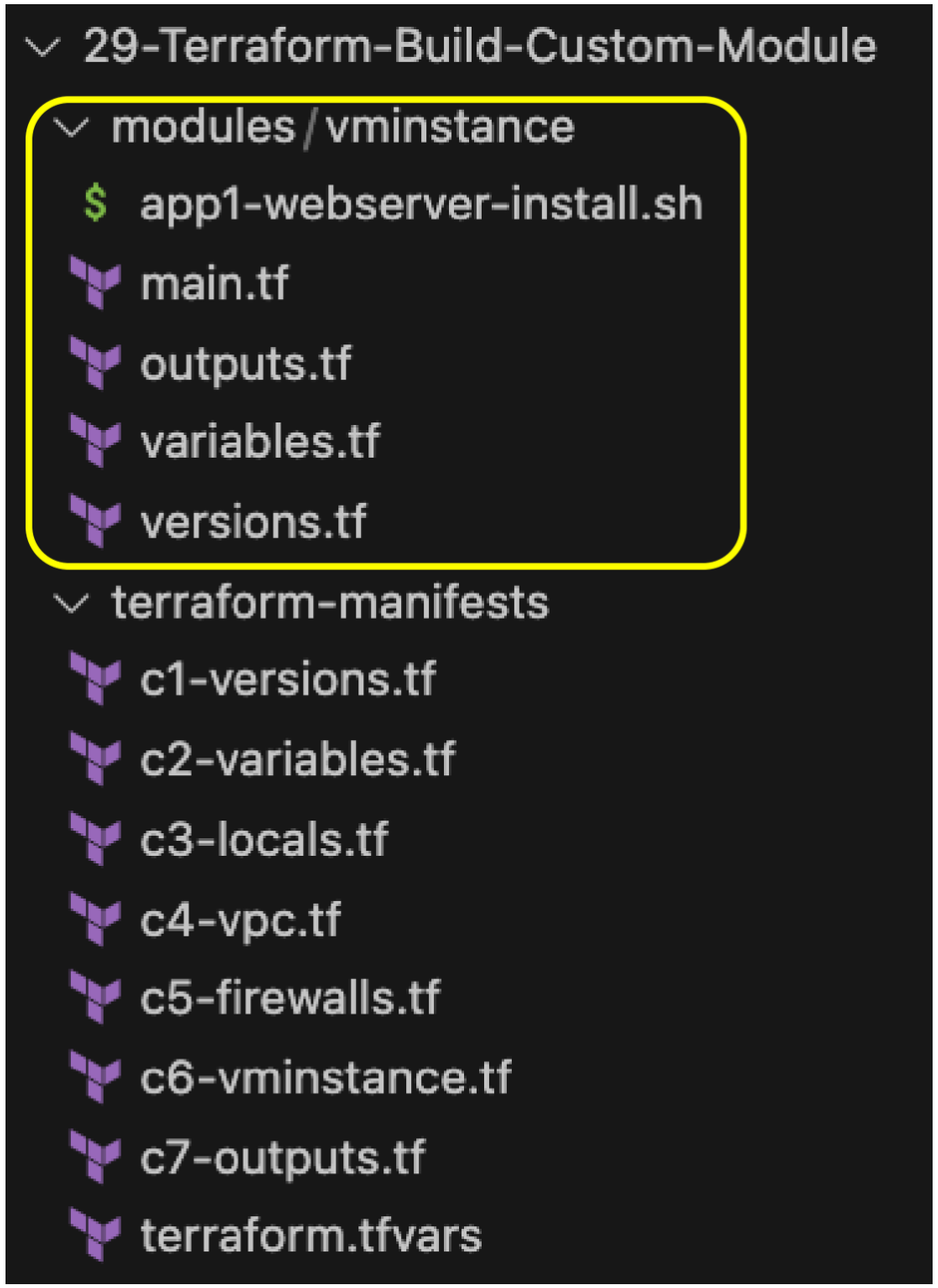
Build a Local Terraform Module and call it from Root Module and Test

Root Module

Call VM Instance child module in Root Module

Child Module





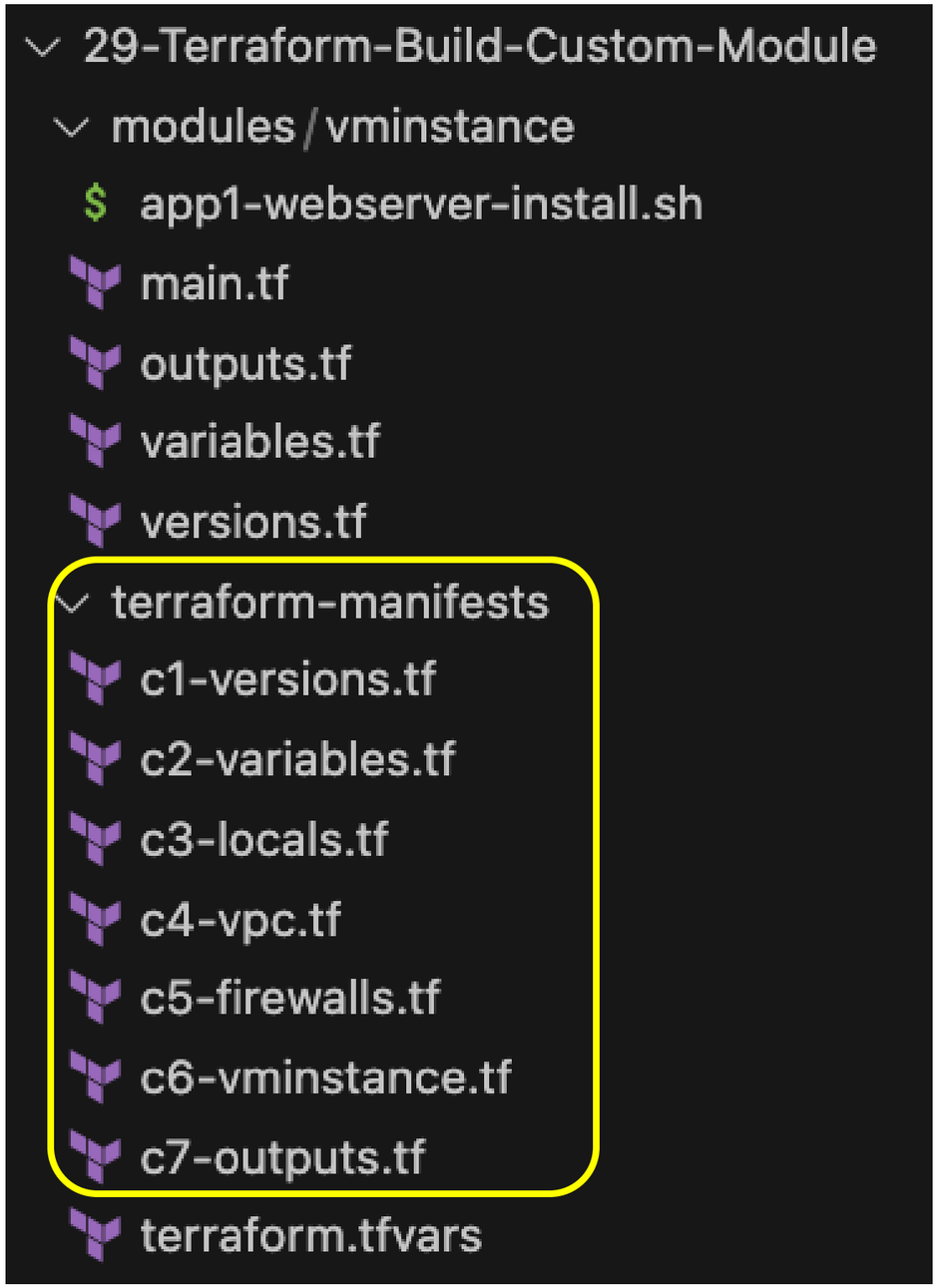
What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Firewall Rules + VM Instance **(Custom or Local Terraform Module)**

Terraform Child Module: VM Instance

versions	Define Terraform Settings block for child or local module
variables	Define Input variables for child or local module
main	Define VM Instance resource for child or local module
outputs	Define outputs for child or local module
App1 Install	Reuse the app1-webserver-install.sh from previous demos



What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Firewall Rules + VM Instance **(Custom or Local Terraform Module)**

Terraform Root Module

NC	No Changes from C1, C2, C3, C5, C5
C6	Define VM Instance module by calling child or local module
C7	Update VM Instance External IP output using VM Instance Module

STACKSIMPLIFY

Demo-30

Terraform

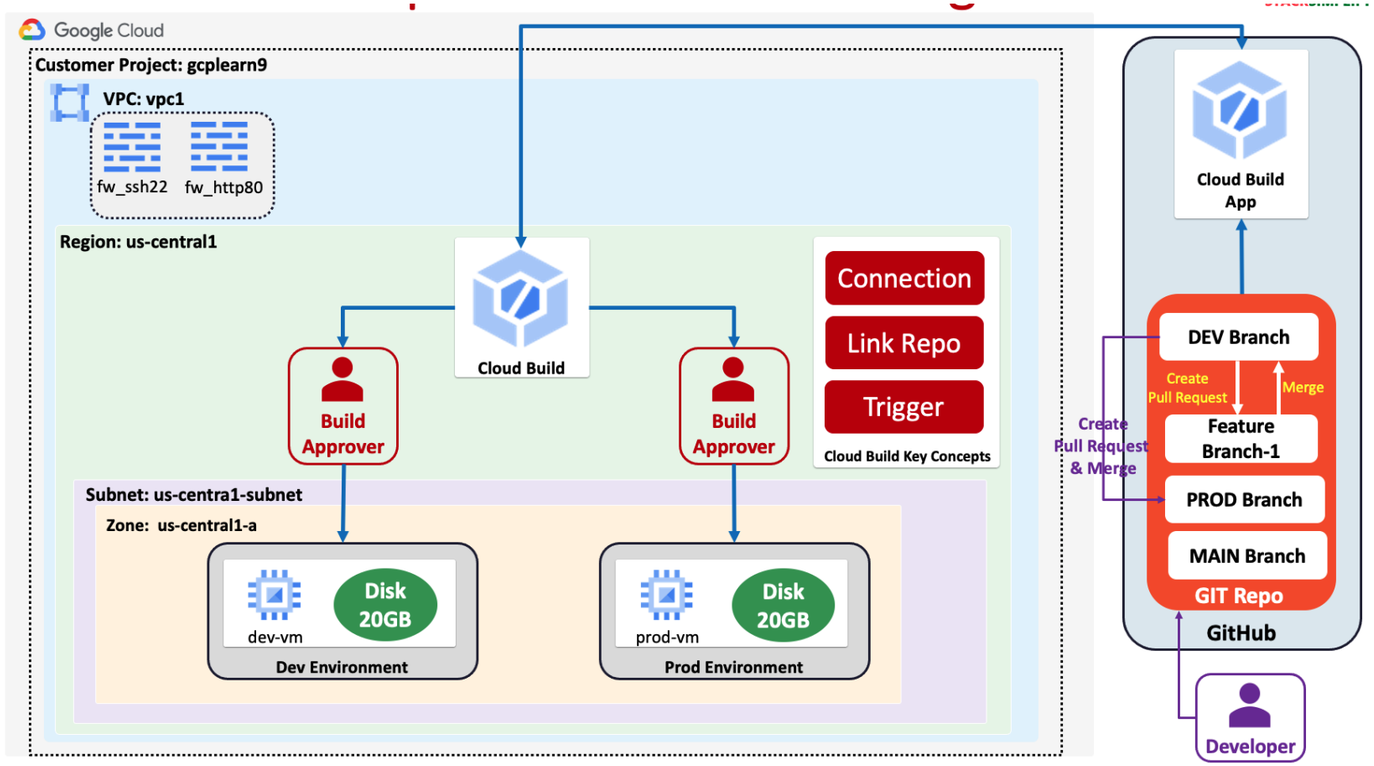
GCP DevOps

GCP Cloud Build, GitHub GCP Cloud Build App and GitHub

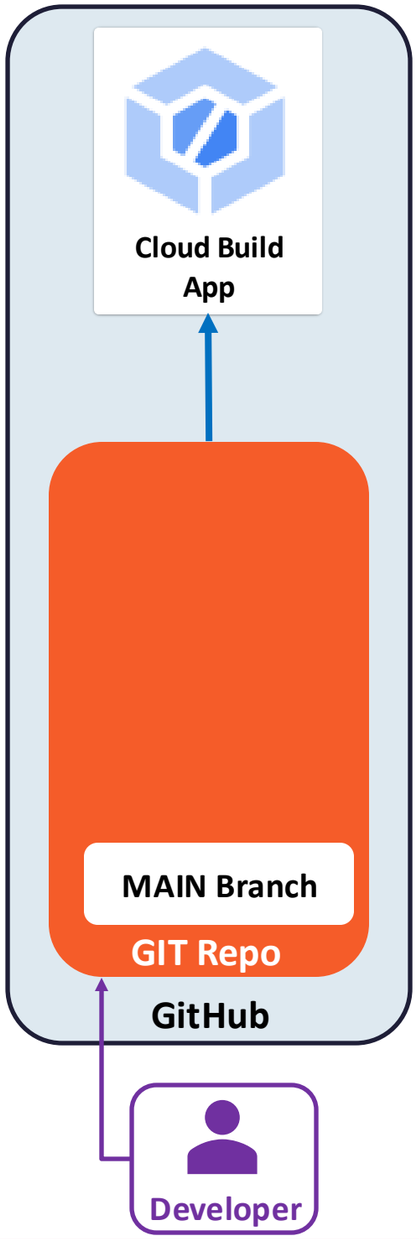
Demo: Custom VPC + Firewall Rules + VM Instance **(Custom or Local Terraform Module)**
Implement DevOps Pipelines for Terraform Configs on GCP

GCP DevOps for Terraform Configs

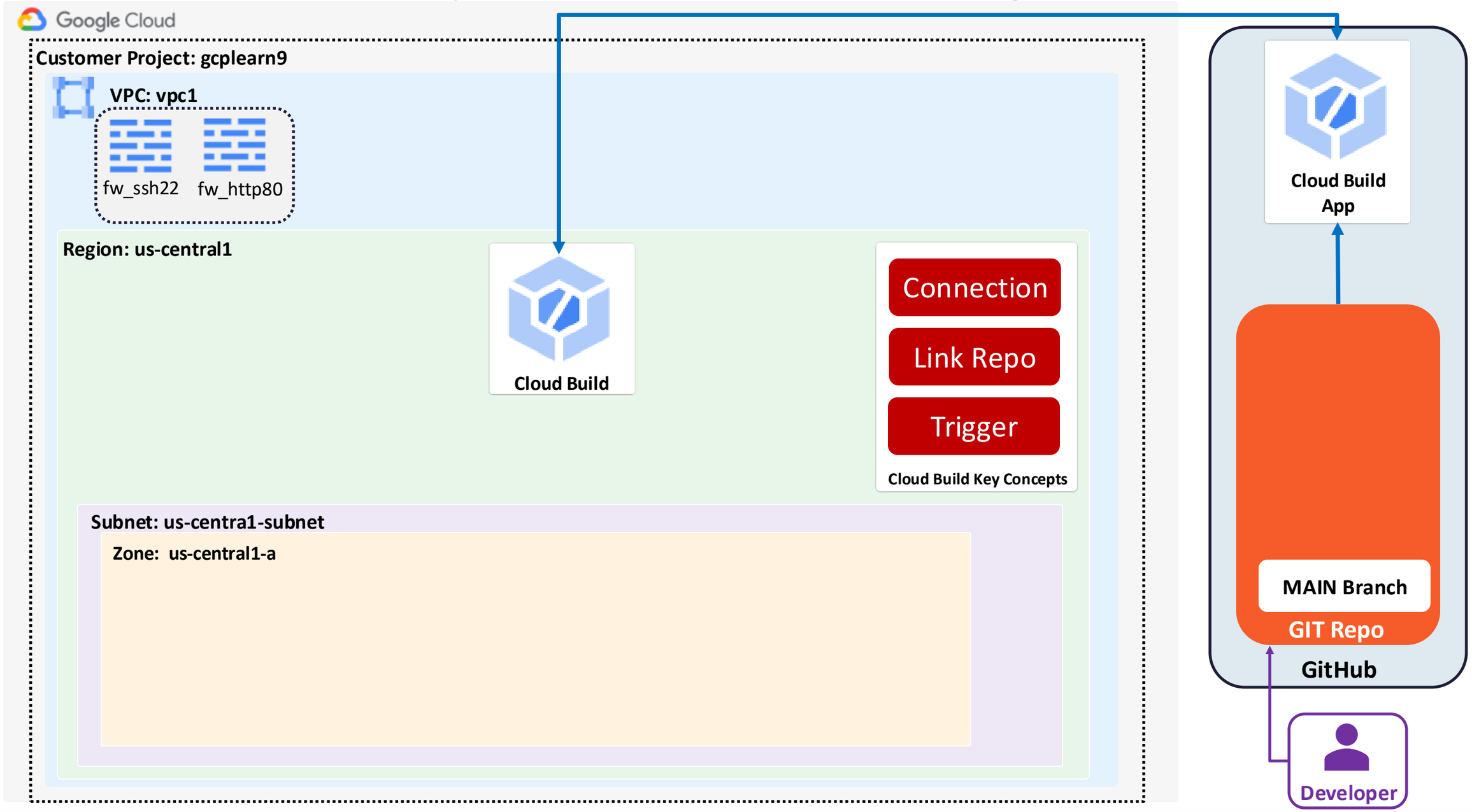
- **The following resources are used for this implementation**
- **GitHub Repositories**
  - GitHub Branches
    - Main branch
    - Dev branch
    - Prod branch
    - Feature branch
  - GitHub Pull Request and Merge concepts
  - GCP Cloud Build App from GitHub Marketplace
- **GCP Cloud Build**
  - Connections
  - Link Repositories
  - Triggers
- **GCP Resources**
  - GCP VPC
  - GCP Firewall Rules
  - GCP VM Instances
- **Terraform Concepts**
  - Terraform Resources
  - Terraform custom modules
  - Terraform Remote Backend using Google Cloud Storage Bucket



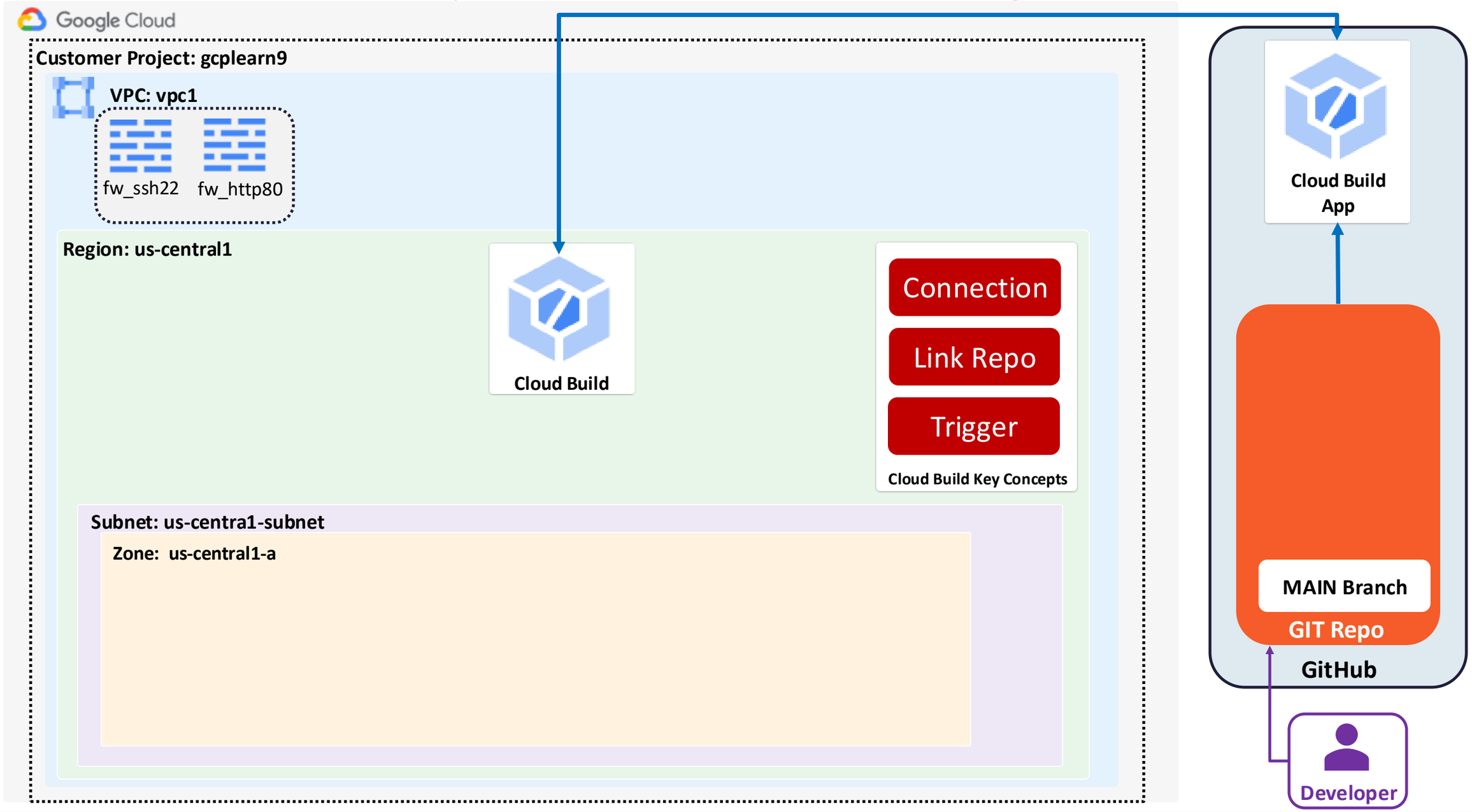
GCP DevOps for Terraform Configs



GCP DevOps for Terraform Configs

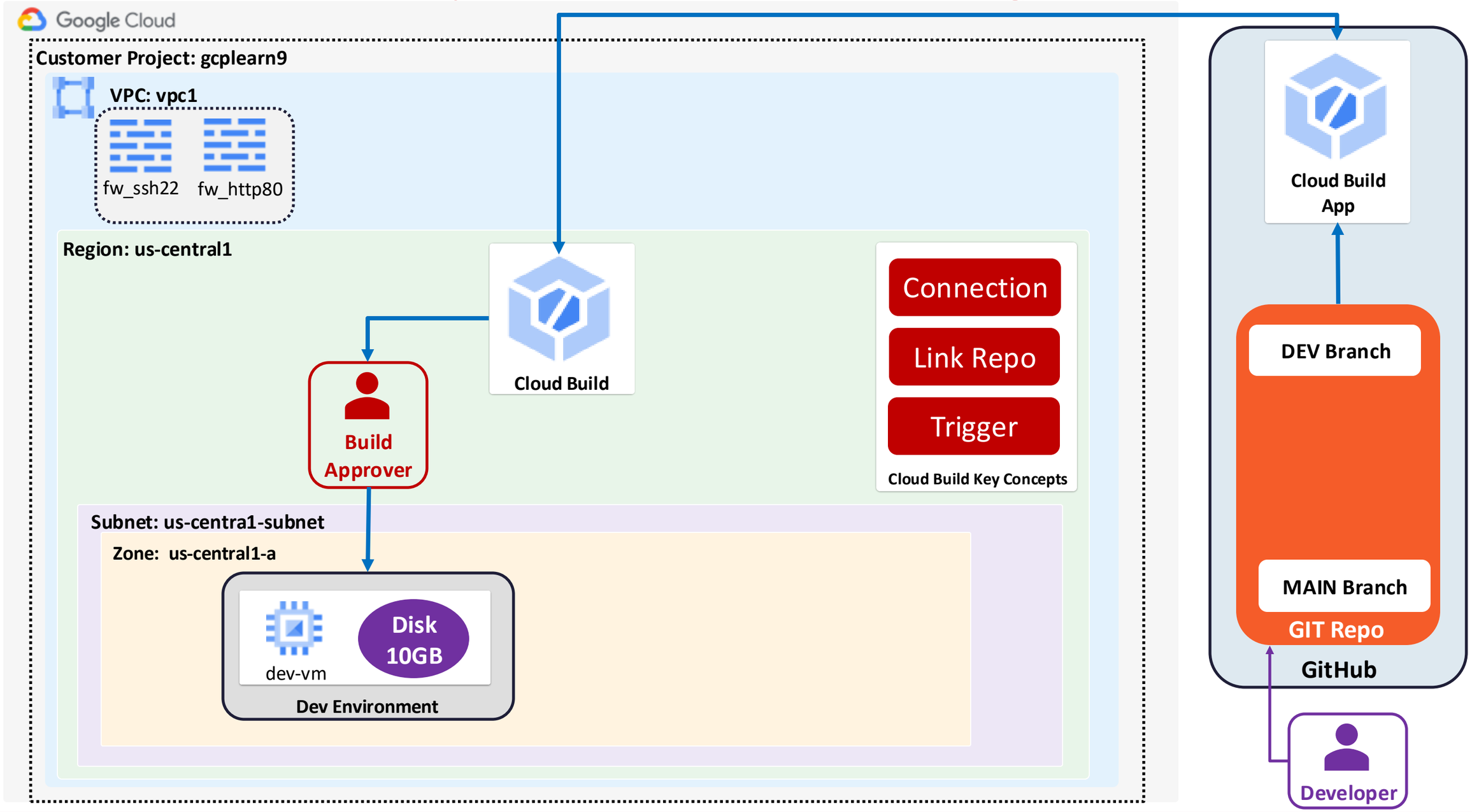


GCP DevOps for Terraform Configs

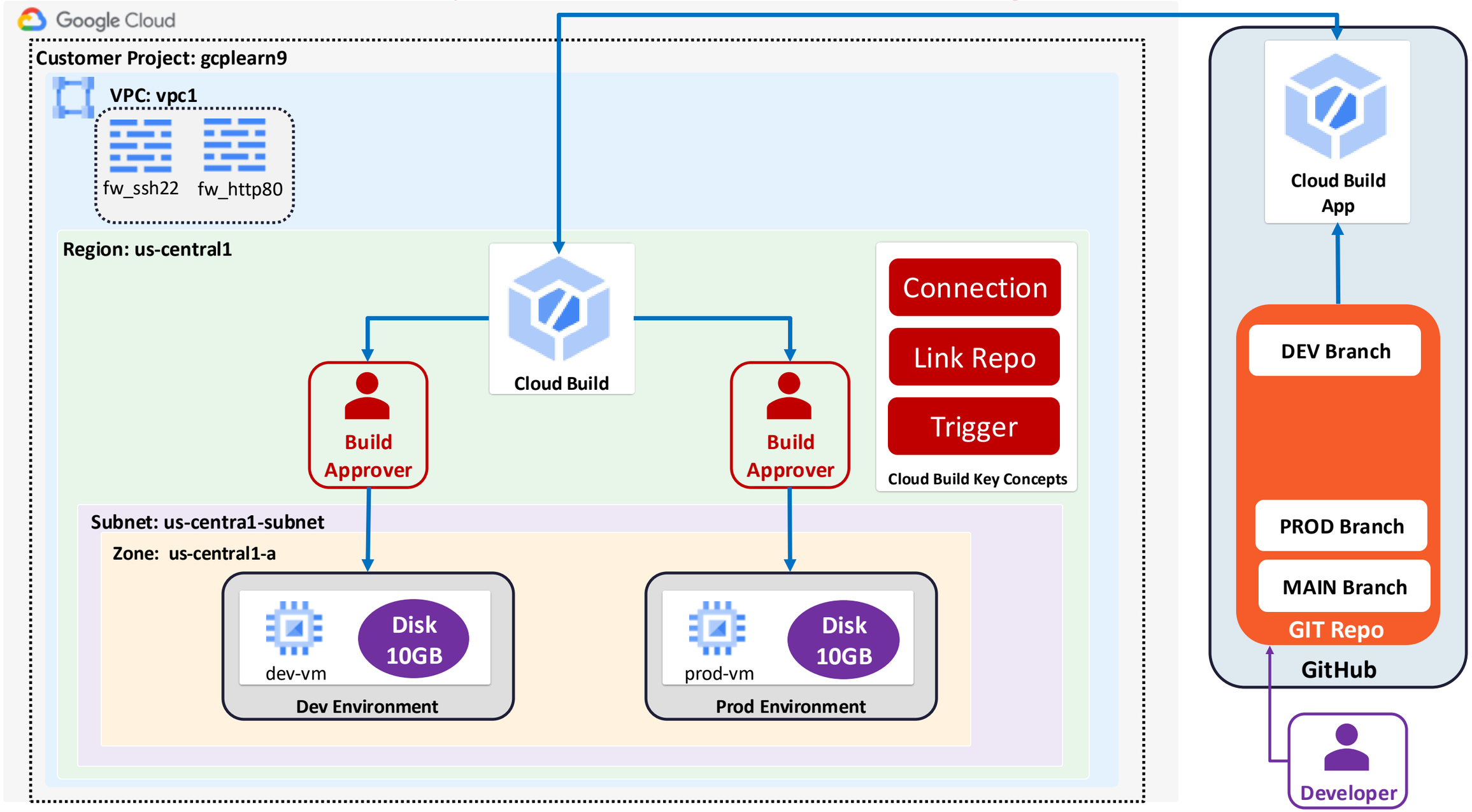


GCP DevOps for Terraform Configs

STACKSIMPLIFY

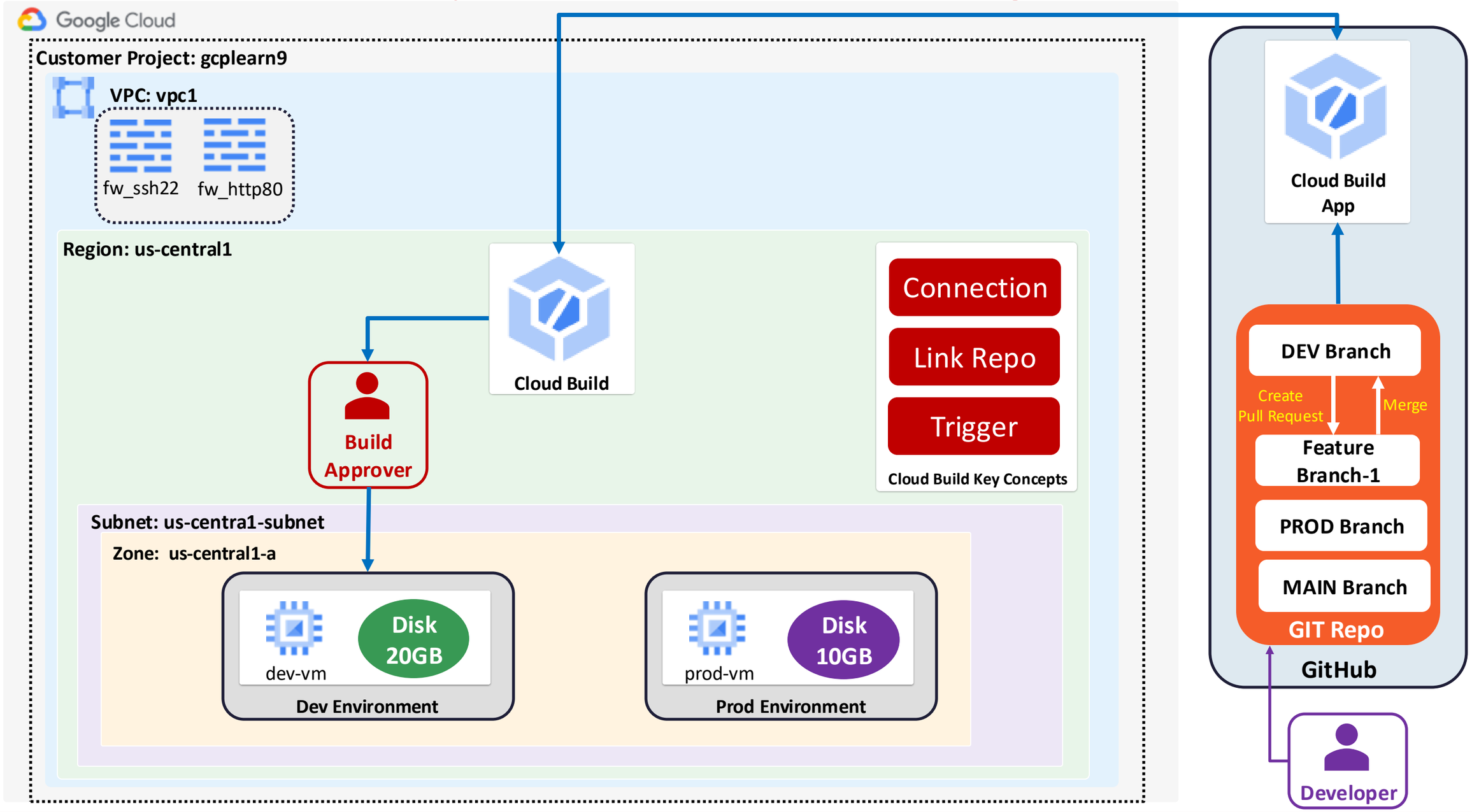


GCP DevOps for Terraform Configs

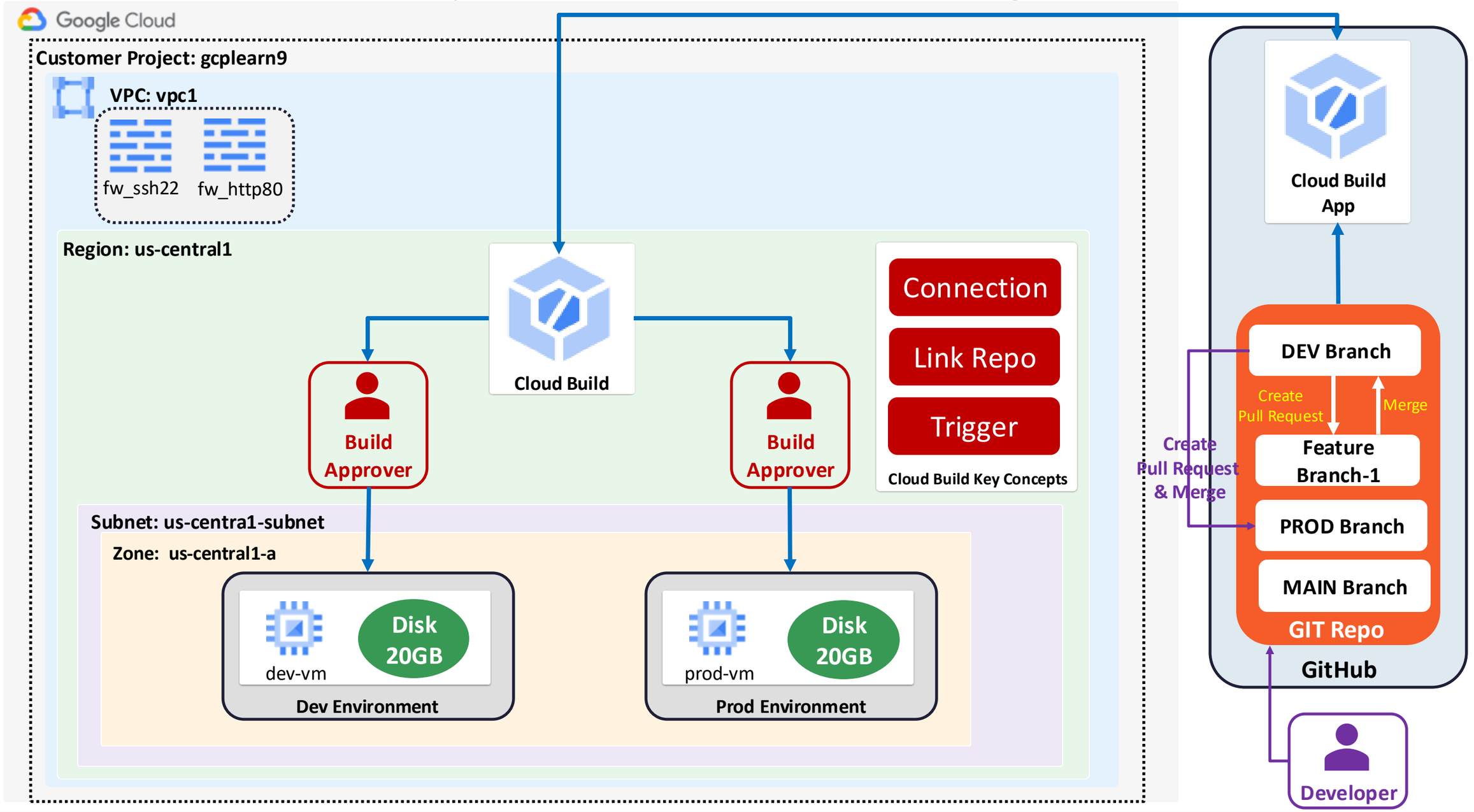


GCP DevOps for Terraform Configs

STACKSIMPLIFY



GCP DevOps for Terraform Configs



What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Firewall Rules + VM Instance (Custom or Local Terraform Module)
Implement DevOps Pipelines for Terraform Configs on GCP

All these TF Configs are copy from Root Module Demo-29 (terraform-manifests folder)

DEV	**Development Environment**
C1	Update GCS Bucket (your bucket name) and State file prefix as “env/dev”
TFVARS	Ensure (environment = "dev”) in terraform.tfvars file
PROD	**Production Environment**
C1	Update GCS Bucket (your bucket name) and State file prefix as “env/prod”
TFVARS	Ensure (environment = ”prod”) in terraform.tfvars file

What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Firewall Rules + VM Instance (Custom or Local Terraform Module)
Implement DevOps Pipelines for Terraform Configs on GCP

All these TF Configs are copy from Child Module Demo-29 (modules folder)

CHILD MODULE

VM Instance Module we have created in Demo-29

cloud build

cloudbuild.yaml contains Terraform commands with DevOps pipeline code

STACKSIMPLIFY

Demo-31

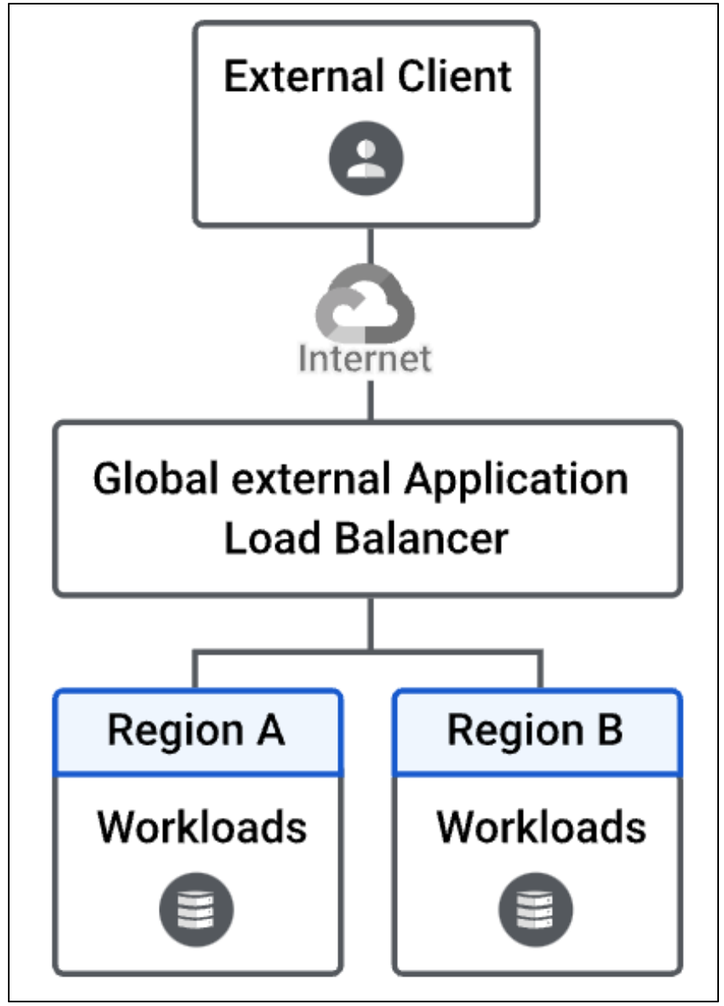
Terraform

Global Application Load Balancer

Demo: Custom VPC + Firewall Rules + Managed Instance Groups + **Global Application Load Balancer**

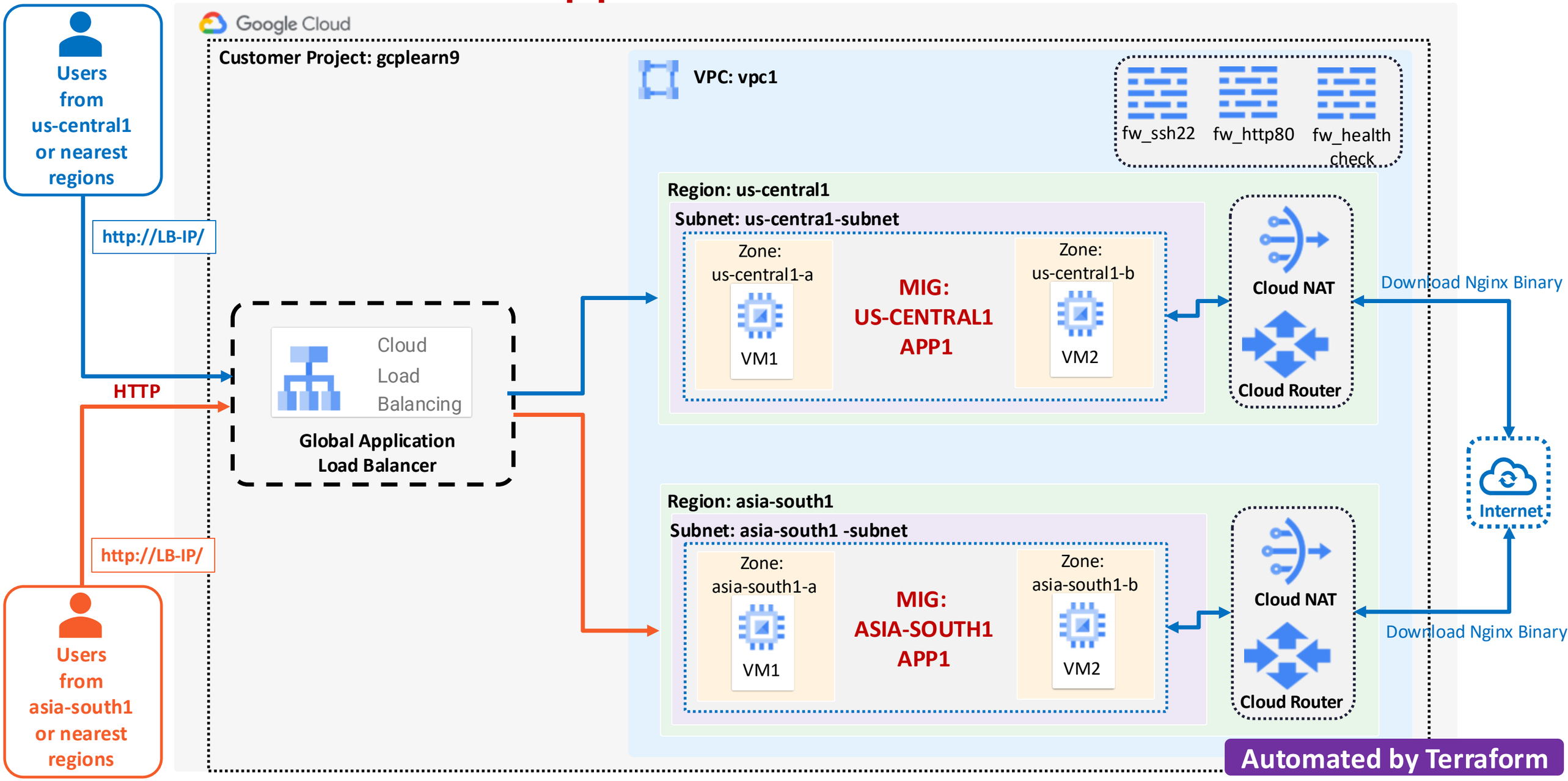
GCP Global Application Load Balancer

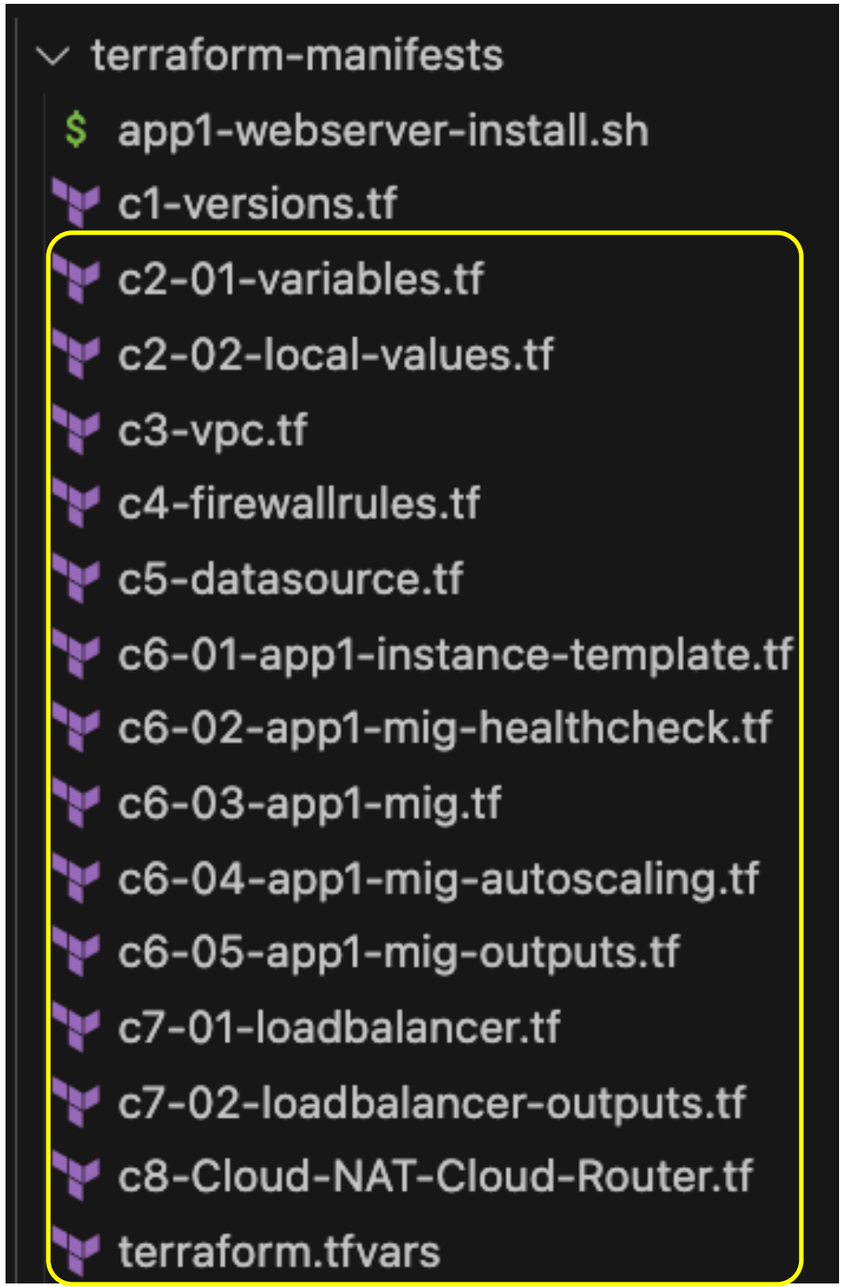
- **Global Application Load Balancer:** Load balances traffic between regions (multi-region load balancing)
- **Load Balancers Single anycast IP Address Feature**
  - Cross-region load balancing
  - Automatic multi-region failover (sends traffic to failover backends if primary backends are unhealthy)
  - Directs traffic to the closest healthy backend
- **Seamless Autoscaling (No config needed - Inbuilt):**
  - can scale as your users and traffic grow
  - It can divert traffic to other regions in the world that can take traffic (if one region is overloaded)
  - Autoscaling does not require pre-warming: you can scale from zero to full traffic in a matter of seconds.
  - In short, it is a FULLY MANAGED SERVICE



GCP Global Application Load Balancer

STACKSIMPLIFY



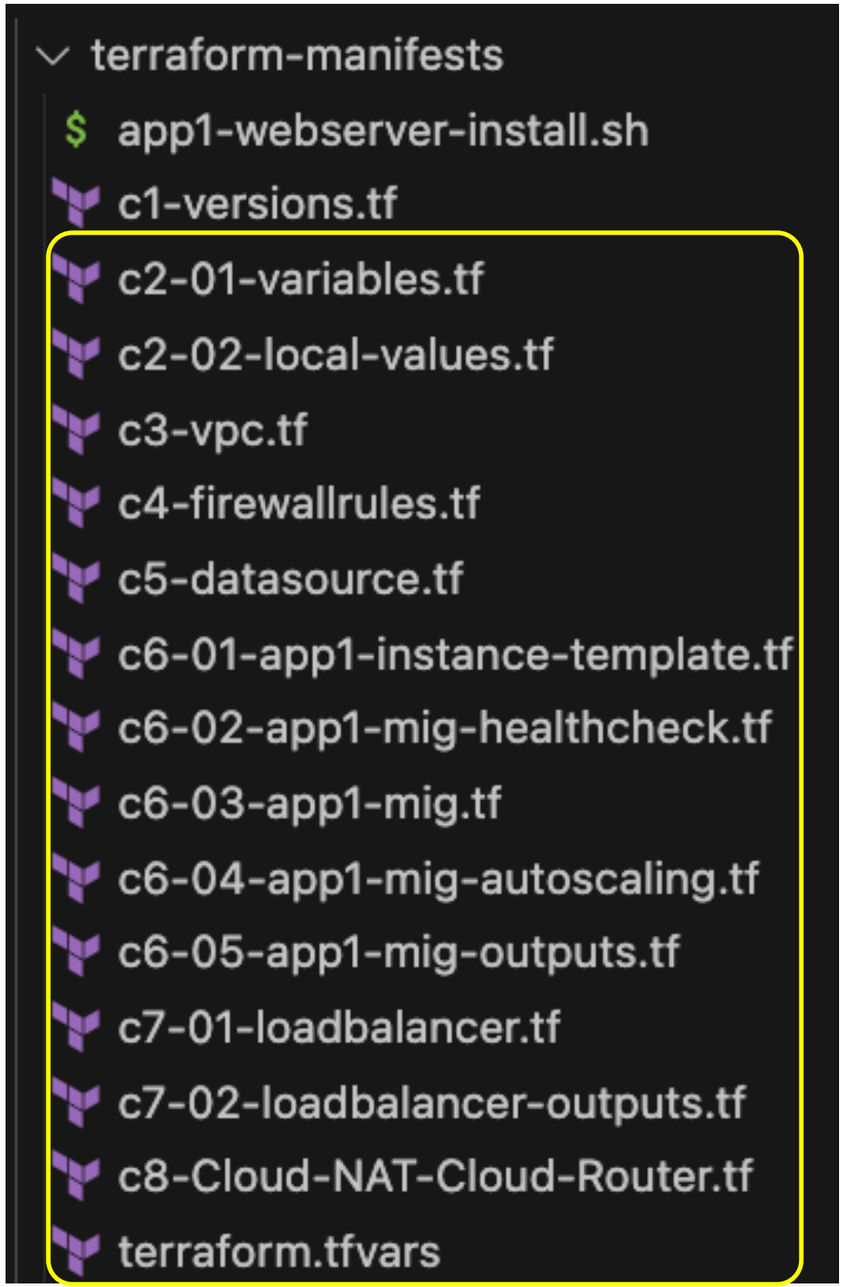


What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Firewall Rules + Managed Instance Groups + Global Application Load Balancer

C2	Define GCP Region-2 variable and update in terraform.tfvars
C3	Create Region-2 Subnet
C5	Create Region-2 Zones data source
C6-01	Create Region-2 Instance Template
C6-02	Create Region-2 MIG Health Check
C6-03	Create Region-2 MIG

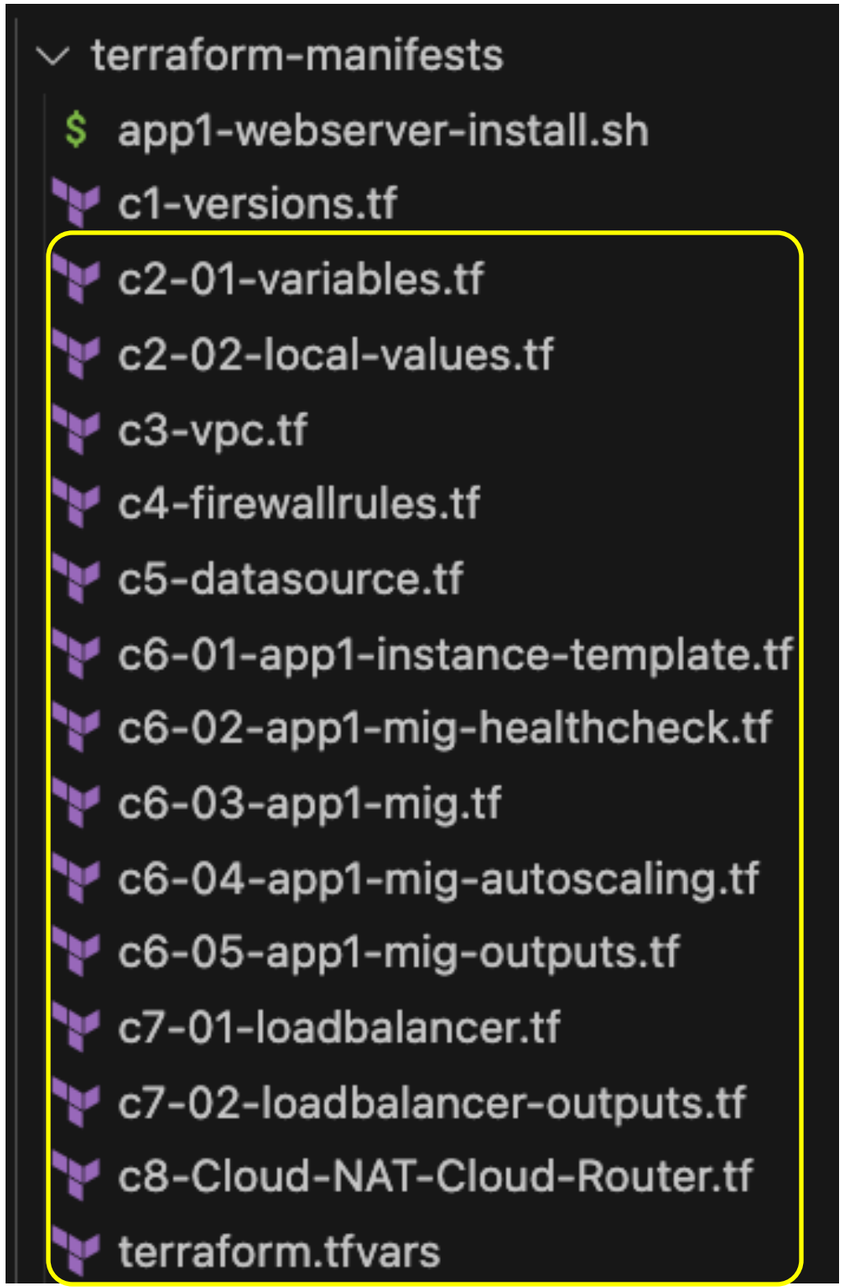


What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Firewall Rules + Managed Instance Groups + Global Application Load Balancer

C6-04	Create Region-2 Autoscaling resource
C6-05	Create Region-2 MIG outputs
C7-01	Create Global Static IP Address for Load Balancer Forwarding Rule
C7-01	Create Global Health Check for Load Balancer
C7-01	Create Global Backend service for load balancer with two regional backends in it
C7-01	Create Global URL Map for Load Balancer



What are we going to learn?

STACKSIMPLIFY

Demo: Custom VPC + Firewall Rules + Managed Instance Groups + **Global Application Load Balancer**

- C7-01 — Create Global HTTP Proxy for Load Balancer
- C7-01 — Create Global Forwarding Rule for Load Balancer
- C7-02 — Create Load Balancer Outputs
- C8 — Create Region-2 Cloud Router and Cloud NAT

STACKSIMPLIFY

ANYTHING AFTER THIS SLIDE IS NON-LIVE SLIDES